HZ BOOKS
华章IT

智能系统与技术丛书

自然言語処理と深層学習 C言語によるシミュレーション

自然语言处理与深度学习

通过C语言模拟

[日] 小高知宏 著
申富饶 于傫 译

机械工业出版社
China Machine Press

作者简介

小高知宏 日本福井大学工学研究科教授。其主要著作有日本欧姆社出版的《从基础开始学会TCP/IP Java网络程序设计 第2版》《初学AI程序设计——用C语言制作人工智能和人工无能》《初学机器学习》《基于AI的大规模数据处理入门》等。

智能系统与技术丛书

自然言語処理と深層学習 C言語によるシミュレーション

自然语言处理与深度学习

通过C语言模拟

[日] 小高知宏 著

申富饶 于傫 译

机械工业出版社
China Machine Press

图书在版编目（CIP）数据

自然语言处理与深度学习：通过 C 语言模拟 /（日）小高知宏著；申富饶，于僖译 . —北京：机械工业出版社，2017.10
（智能系统与技术丛书）

ISBN 978-7-111-58657-9

I. 自… II. ① 小… ② 申… ③ 于… III. 自然语言处理—研究 IV. TP391

中国版本图书馆 CIP 数据核字（2017）第 295905 号

本书版权登记号：图字 01-2017-7497

自然语言处理与深度学习：通过 C 语言模拟

出版发行：机械工业出版社（北京市西城区百万庄大街 22 号 邮政编码：100037）
责任编辑：吴晋瑜
责任校对：殷 虹
印 刷：中国电影出版社印刷厂
版 次：2018 年 1 月第 1 版第 1 次印刷
开 本：186mm × 240mm 1/16
印 张：11.5
书 号：ISBN 978-7-111-58657-9
定 价：49.00 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换
客服热线：（010）88379426 88361066
投稿热线：（010）88379604
购书热线：（010）68326294 88379649 68995259
读者信箱：hzit@hzbook.com

THE TRANSLATOR'S WORDS

译 者 序

这是一本自然语言处理与深度学习的入门书，适合对自然语言处理与深度学习感兴趣的初学者阅读。书中不涉及高深的数学理论，读者仅需要掌握C语言的基础知识和高中程度的数学知识即可进行学习。本书也可供相关专业人士参考。

译者在Microsoft Visual Studio Professional 2013下编译执行了本书涉及的所有程序和实例，书中给出的结果为译者实际执行的结果，和原书略有不同，但都是正确的。读者可以从如下网址下载所有程序代码和实例的数据：http://cs.nju.edu.cn/rinc/book2017。

因为本书涉及的实例数据是用Shift_JIS汉字编码表示的日语文本，在中文环境下测试时需要将“非Unicode程序的语言”调整为“日语（日本）”，即做如下操作：

“控制面板”→“时钟、语言和区域”→“区域和语言”→“管理”→“更改系统区域设置”→日语（日本）”。

在完成测试后，可以按上述方法将“非Unicode程序的语言”调整回“中文（简体，中国）”。即便执行结果和书中报告的结果不一致，也是正确的，因为神经网络中用随机数来初始化权重，每次执行可能会带来不同的结果。

PREFACE

前　言

深度学习技术在计算机图像识别领域取得了重大成果，这一技术目前已经逐渐应用于机器学习的多个不同领域，使人工智能发展到了过去所不能达到的能力层次。同样，深度学习也能应用于自然语言处理领域，能够解决过去不能处理的各种自然语言处理问题。

本书初步探索了将深度学习应用于自然语言处理的方法，概述了自然语言处理的常见概念，通过具体实例说明了如何提取自然语言文本的特征以及如何考虑上下文关系来生成文本。本书中，自然语言文本的特征提取是通过卷积神经网络来实现的，根据上下文关系来生成文本则利用了循环神经网络。这两个网络是深度学习领域中常用的基础技术。

本书通过实现C语言程序来具体讲解自然语言处理与深度学习的相关技术，所给出的程序都能在普通的个人电脑上执行。通过实际执行这些C语言程序，确认其运行过程，并根据需要对程序进行修改，读者能够更深刻地理解自然语言处理与深度学习技术。

本书的完成离不开作者在福井大学从事科研活动积累的经验，在此特别感谢提供这样机会的福井大学教职员和学生。此外，特别感谢Ohmsha出版社提供了出版本书的机会。最后，感谢支持我完成本书的家人洋子、研太郎、桃子以及优。

小高知宏

2017年2月

CONTENTS

目　录

CHAPTER 1

第 1 章

自然语言处理与深度学习

本章首先介绍自然语言处理，概述为了实现自然语言处理所需要做的研究，然后通过若干例子介绍深度学习这一机器学习领域中的方法，最后基于上述讨论说明自然语言处理与深度学习之间的关系。

1.1 自然语言处理

1.1.1 什么是自然语言处理

自然语言处理（natural language processing）是指利用计算机程序对自然语言进行处理的技术。这里的“自然语言”是指像日语、英语这样的作为人们交流和思考工具的语言。自然语言处理中的“处理”是指数据的检索、整理、保存等基本的数据处理，以及语义提取和不同语言间的翻译等基于自然语言特性的高级处理。

图 1.1 给出了自然语言处理的具体应用实例。

自然语言处理中最基本且实用的技术是对自然语言文本进行输入和编辑的辅助技术（图 1.2）。例如，日语输入中辅助进行平假名和汉字转换、日语字处理机的编辑辅助功能等都是通过研究自然语言处理技术获得的。此外，随着智能手机的普及，通过语音输入自然语言文字的技术也已经得以普及。目前，上述这些很大程度上已经实现的自然语言

处理技术已经得到了广泛应用。

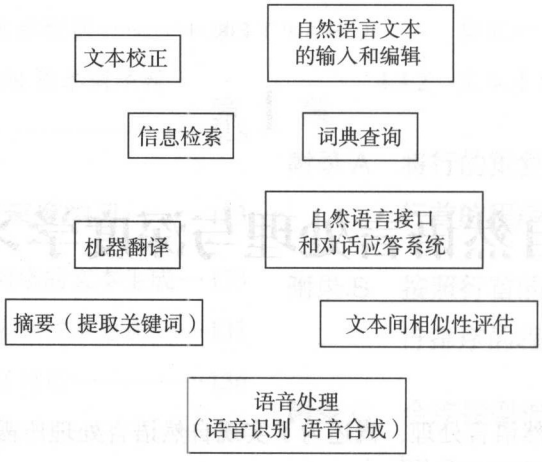


图 1.1 自然语言处理的具体应用实例

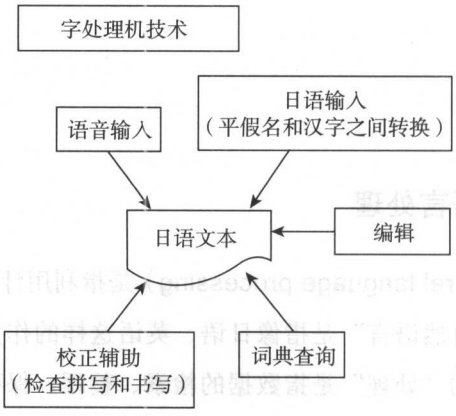


图 1.2 自然语言文本的输入和编辑

日语字处理机不仅具有文本编辑功能，也集成了辅助进行文本生成、文本校正等多种技术。例如，用于调查词语用法的词典检索功能、指出拼写错误及检查书写不规范的文本校正功能等，都是自然语言处理技术的应用实例。

根据自然语言进行**信息检索**是一种典型的互联网应用。在**搜索引擎**网站中输入要搜索的单词或短语，便能检索出和输入单词或短语相关的信息。在搜索时，不仅能检索出

和输入单词完全一致的信息，也能搜索出和检索单词大体上相似的模糊内容。有时还可以修正搜索对象单词的书写错误、补全正在输入的或已经输入完毕的检索词，这些也都是自然语言处理技术的应用实例（图 1.3）。

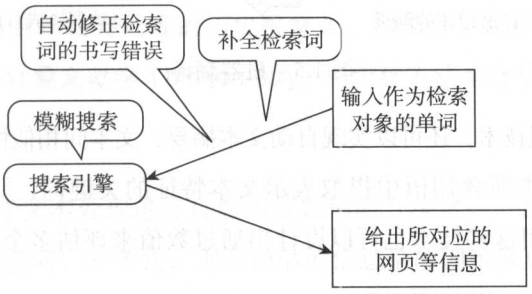


图 1.3　通过自然语言进行信息检索

自然语言接口能够通过和机器对话来操控机器，智能手机的对话应答系统是一个实用化的实例。这一接口能够通过语音输入自然语言文本、根据输入进行检索以及将输入结果保存成文本并进行编辑。此外，也可以通过语音给出指令以操控系统（图 1.4）。

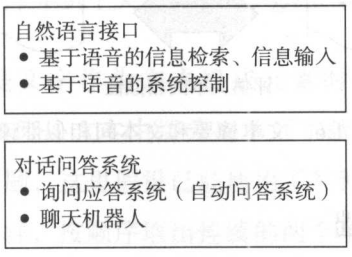


图 1.4　自然语言接口

在对话系统中，有一类对话应答系统不用于信息检索等特定目的，其系统目标是持续进行无特定内容而漫不经心的对话，这类对话应答系统称为**人工无能**或**聊天机器人**（chatbot）。这类系统也是随着自然语言处理技术的应用而诞生的。

自然语言处理技术的另一个应用实例是**机器翻译**（machine translation）技术。机器翻译是指在不同语言文字间进行相互变换，这一技术也逐渐得到了实用化（图 1.5）。在已经实用化的系统中，除了面向特定领域中固定格式文本的翻译系统之外，也有面向一般不固定格式文本进行翻译的机器翻译软件系统。例如，作为一种网络服务而实现的网

页机器翻译系统就可以将非日语记述的网页翻译成日语表示出来。

图 1.5 机器翻译

应用自然语言处理技术，还可以实现自动文本摘要、文本间相似性评估等任务（图 1.6）。文本摘要会用到从文本所含用语中提取表示文本特征的关键词、写出表达文本意义的文摘等技术，此外，利用这些技术也可以设计出通过数值来评估多个文本间相似性的方法。

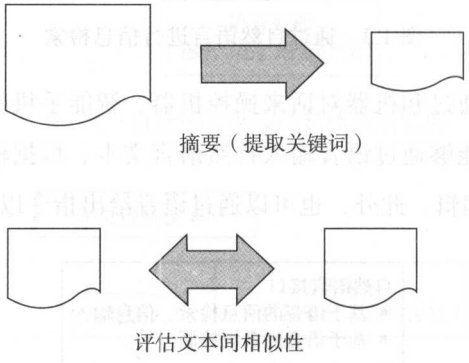


图 1.6 文本摘要和文本间相似性评估

1.1.2 自然语言处理基础

本节将概述进行自然语言处理研究的流程，归纳对于理解自然语言处理技术而言必要的基本事项。

1. 文本处理

文本处理（text processing）是自然语言处理的基本技术，其处理对象是由文字排列而成的文本。文本处理包括分解文本中的文字、提取文字块[⊖]、文字和文字块的分类、检索、计数，以及对上述结果进行统计从而评估文本等处理技术。

⊖ 一个或多个相邻文字组成的语言整体。——译者注

计算机是处理符号的机器。文本可以看作符号的排列，文本处理的处理对象则可以看作由符号的排列组成的文本。因此，文本处理可以理解为一种符合计算机本质操作特性的处理过程。

基于文本处理进行自然语言处理的历史很悠久。例如，早在计算机发明之前就开始研究的计量语言学或称**计量文献学**（bibliometry）用统计方法来评估文献中所含文字的种类、语义等文本特征。

n-gram 分析是基于文本处理进行自然语言处理的方法之一。20 世纪 40 年代，人类发明了电子计算机。紧接着，50 年代，大名鼎鼎的信息论的创始者克劳德・香农（Claude Shannon）提出了 *n*-gram。

n-gram 将连续 *n* 个文本组成元素分解出来进行处理。如图 1.7 所示，将文本“すももももももものうち”中的文字作为单位来生成 *n*-gram。考虑 *n*=3（即 3-gram）的情形，从开头依次取出 3 个文字生成文字块，就形成了本例中关于文字的 3-gram 集合。

文字3-gram示例

すももももももものうち

すもも
ももも
ももも
…
ものう
のうち

图 1.7 *n*-gram 示例（1）

图 1.8 是使用单词而不是文字作为文本组成元素时生成 *n*-gram 的例子。在日语中，如何从文本中切分出单词本就是自然语言处理技术需要研究的课题，这里假设已经使用了某种方法将单词从文本中切分出来了。然后，和前面的例子一样，按顺序取出连续的两个单词，就可以生成关于单词的 2-gram。如图 1.8 所示，以“自然言語を処理する”这一文本为例，生成了关于单词的 2-gram，将文本分割成单词后，将每两个单词按顺序组合起来，就生成了单词的 2-gram。

可以通过文字 *n*-gram 和单词 *n*-gram 来探索文本的特征。例如，对本书开头部分的文本生成文字 3-gram 集合，其结果如图 1.9 所示。图中给出了 3-gram 出现频率的统计排序结果表。在该表的上方能看到“然言语”“自然言”“言语处”“语处理”“系统”等 3-gram，这就说明本书开头部分是在谈论“自然语言处理”“语言处理系统”等话题。

本书第 2 章将讲述用 C 语言程序来生成 *n*-gram 的方法。

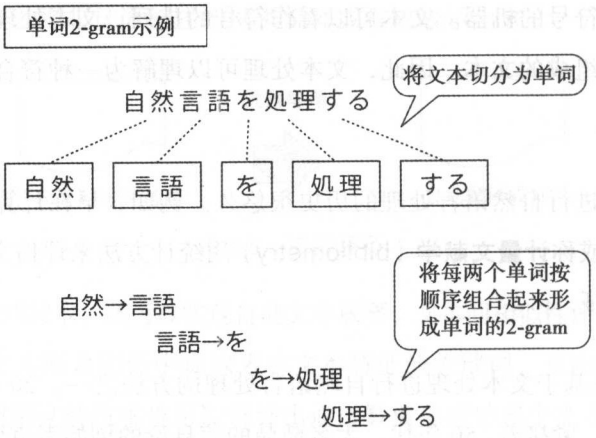


图 1.8 *n*-gram 示例（2）

文字3-gram示例

频度	3-gram
24	然言語
24	自然言
16	ます。
11	言語処
11	語処理
11	システ
11	ステム
10	
9	自然
8	です。
8	ること
8	日本語
7	れてい

（以下省略）

由文字3-gram的出现频率，可知文本中在谈论“自然语言处理”“语言处理系统”等话题

图 1.9 *n*-gram 频率的分析示例（1）

图 1.9 中的示例是对于文字 3-gram 进行分析的，如果用同样的方法对单词 2-gram 进行分析，会发现什么呢？前面已经说过，为了生成单词 2-gram，需要从将要分析的文本中将单词分割切取出来。如果分析对象是英语或者德语，其文本在书写时已经通过空格将单词分开了，因此，对这类语言文本而言，切分单词非常简单。对于如日语这样没有通过空格将单词分开写的语言来说，需要某种自然语言处理技术来将单词切分出来。

从文本中切分出单词、对单词的作用进行分析的处理技术称为**形态素分析（morphological analysis）**。为了生成关于日语单词的 *n*-gram，需要通过形态素分析方法将单词切分出来。第 2 章将讨论如何通过程序来进行形态素分析。

通过形态素分析方法将单词切分出来，生成单词 2-gram 并进行统计，其分析结果如图 1.10 所示。图中的记号“ ->”将 2-gram 的前半部分和后半部分区分开来。例如，频率最高的“自然 -> 言语”表示连续出现的“自然”单词和“言语”单词的 2-gram。从图中的分析结果可以看出，该段文本中所含的“自然 -> 言語”以及“言語 -> 处理”这样的关键词比较多。

单词2-gram示例

频度	2-gram
24	自然->言語
16	ます->。
15	は->、
15	->
11	言語->処理
10	さ->れ
9	->自然
8	です->。
7	1->.
7	図->1
6	て->い
6	する->こと
6	れ->て
6	い->ます
5	類似->性
5	文書->の
5	し->たり
5	機械->翻訳
5	の->類似
5	語->の
5	情報->検索
5	処理->技術
5	応答->システム
5	的->な
5	処理->の
4	言語->インタフェース

由单词2-gram的频率可以发现，文本中所含的“自然->言語”“言語->处理”等关键词较多

（以下省略）

图 1.10　*n*-gram 频率的分析示例（2）

n-gram 不仅能对给定文本进行分析，也能用于生成新文本。如果将图 1.10 所示的结果作为生成文本的规则使用，就可以从某个单词开始，根据单词之间的连续关系来生成文本。

例如，尝试从“自然”这一单词开始生成文本。根据图 1.10 的结果，“自然”单词和“言語”单词是连接在一起的。再看“言語”，它和“处理”连接在一起的频率为 11。此外，“言語”还和“接口”这一单词以频率 4 连接在一起。在“言語”后面到底选择哪一个单词，可由另一种规则来决定。如果选择了“处理”，下面就是“技術”或者“の”和它连接在一起。图 1.10 中省略了频率比较低的部分，实际上，“处理”也偶尔和一些“する”“とともに”“を”等连接在一起。按照上述分析，可以通过图 1.11 所示的方式生成文本。

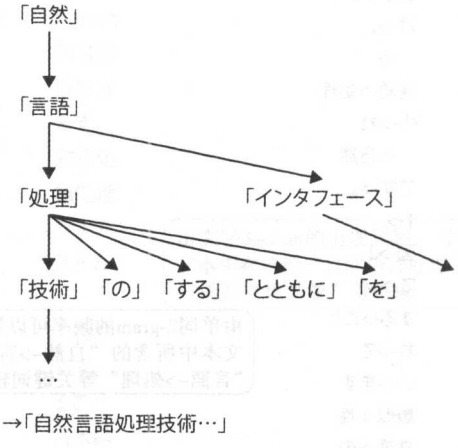


图 1.11 用 *n*-gram 规则生成文本的示例

2. 形式文法

在学习自然语言时，除了要记忆所学语言的单词外，还需要学习该语言的语法。例如，小学生和中学生刚开始学习英语时，不仅要记住英语单词，还要学习英语的语法。不仅学习外文时需要，学习古文时也需要学习句法和词性等语法知识。事实上，即使在学习现代日语时也需要学习语法。因此，在自然语言处理中，对语法进行处理是非常必要的。

从自然语言处理的历史来看，由 Noam Chomsky 提出的**形式语言理论**（formal language theory）可看作自然语言处理的基础语法理论。Chomsky 于 20 世纪 50 年代提出了作为形式语言理论的**生成语法**（generative grammar），此后形式语言理论得到了充分发展。**短语结构文法**（phrase structure grammar）是生成语法中具有代表性的例子，这里以此为例进行说明。

短语结构文法是将某个符号用其他符号替换来形成重写规则集合，从而表示文本的构成方法。图 1.12 给出了一个重写规则的示例。

```
<文本>→<名词句><动词句>……重写规则①
<名词句>→<名词>……重写规则②
<名词句>→<形容词><名词句>……重写规则③
<动词句>→<动词>……重写规则④
<名词>→我……重写规则⑤
<动词>→步行……重写规则⑥
<形容词>→美丽的……重写规则⑦
<形容词>→凛凛的……重写规则⑧
```

图 1.12　短语结构文法中重写规则示例

在图 1.12 中，用尖括号 <> 括起来的表示文本中具有语法作用的构成要素，这些符号称为**非终止符**（nonterminal symbol）。尖括号 <> 没有括起来的称为**终止符**（terminal symbol）。在以语法作为记叙对象的语言中，终止符是实际上构成文本的单词。

短语结构文法中，箭头的两边配置了相应的符号，按顺序从左向右进行重写，就可以生成实际的文本。重写是从特定的非终止符开始的，可特别用**起始符**（start symbol）来称呼这一非终止符。在图 1.12 的例子中，<文本> 就是起始符。

图 1.12 中最开始的重写规则①是指起始符为 <文本>，这一符号可通过 <名词句><动词句> 这两个非终止符的排列来重写。将这一规则和规则②、规则④一起应用，则 <文本> 可通过 <名词><动词> 的排列来重写。进一步应用规则⑤和规则⑥，可以生成“我步行”这样的文本（图 1.13）。像这样，短语结构文法中的重写规则就给出了生成文本的规则。

生成“我步行”

<文本>→<名词句><动词句>……使用规则①
→<名词><动词句>……使用规则②
→<名词><动词>……使用规则④
→我<动词>……使用规则⑤
→我步行……使用规则⑥

图 1.13 重写规则的应用示例（1）

如果将规则的应用方法改变一下，就可以生成其他文本。例如，在图 1.14 中，起始符是<文本>，生成了“凛凛的美丽的美丽的我步行”这一文本。

这里反复使用的规则③中，将<名词句>通过<形容词><名词句>这两个非终止符的排列来进行重写。在该规则中，箭头的左边和右边出现了同样的符号<名词句>。因而，反复应用这样的循环规则，<名词句>前面的<形容词>会不断增加，导致非终止符的排列不断变长。

生成“凛凛的美丽的美丽的我步行”

<文本>→<名词句><动词句>……使用规则①
→<形容词><名词句><动词句>……使用规则③
→<形容词><形容词><名词句><动词句>……使用规则③
→<形容词><形容词><形容词><名词句><动词句>……使用规则③
→<形容词><形容词><形容词><名词><动词句>……使用规则②
→<形容词><形容词><形容词><名词><动词>……使用规则④
→凛凛的美丽的美丽的我步行……使用规则⑤⑥⑦⑧

图 1.14 重写规则的应用示例（2）

重写规则不仅能应用于文本生成，还可以用于**句法分析（syntax analysis）**，即可以用于调查给定文本的句子结构。例如，在对“美丽的我步行”这一例文进行分析时，可通过图 1.12 中的重写规则来调查其句子的结构，从起始符<文本>出发，依次对照相应的规则，搜索能够适用于给定文本的规则，其搜索过程如图 1.15 所示。

将图 1.15 中成功应用的规则进行整理，可得到如图 1.16 所示的结果，这样的数据结构称为**句法树（syntax tree）**。得到句法树后，结合文本的结构和单词的意义，可以了解

文本全文的意义。

< 文本 > → < 名词句 >< 动词句 > ……使用规则①
→ < 名词 >< 动词句 > ……使用规则②
例文“美丽的我步行”的开头部分和 < 名词 > 不一致，应用规则②失败，退回一步并使用②以外的规则
→ < 形容词 >< 名词句 >< 动词句 > ……使用规则③
→ < 形容词 >< 名词 >< 动词句 > ……使用规则②
→美丽的 < 名词 >< 动词句 > ……使用规则⑦
→美丽的我 < 动词句 > ……使用规则⑤
→美丽的我 < 动词 > ……使用规则④
→美丽的我步行 ……使用规则⑥

图 1.15 基于重写规则进行句法分析

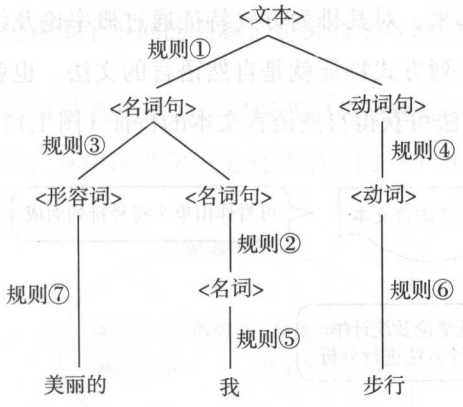


图 1.16 句法树示例

短语结构文法是由非终止符、终止符、重写规则以及起始符这四要素所规定的语法。这里所给出的示例非常简单，但如果重写规则足够多，就能够表示复杂文本的句法。关键问题在于，怎样才能得到重写规则。

历史上大多以人工方式对文法进行分析并根据其分析结果记录下重写规则。然而，自然语言非常复杂，乍一看就会发现大量不同的语法事项。因此，通过人工方式记录重写规则需要繁重的劳力。而且，非常难以验证庞大的规则集合作为一个整体是否正确、各部分之间是否矛盾。

这样一来，通过人工方式来记录重写规则，对于自然语言这一领域而言是非常困难的。因此，不存在利用人工方式构建句法规则来建立一般处理系统的例子。

3. 统计自然语言处理与机器学习、深度学习

记录文法和构建词语字典所需处理的信息量非常庞大，要花费大量工夫，且处理内容本身也是需要专门知识的难题。因此，如果能以自动化方式代替人工方式来完成上述任务，就可以解决上述问题。

一种自动化方式是基于统计的自然语言处理技术。统计学给人的第一印象是以数值为处理对象的，**统计自然语言处理**（statistical natural language processing）则以自然语言为处理对象并探寻其统计性质。统计自然语言处理是指，将作为输入信息的自然语言文本以符号形式排列起来，对其排列方式特征通过概率论及统计学等统计方法进行分析。这样所得到的符号排列方式特征就是自然语言的文法。也就是说，在不需要人工参与的情况下，利用统计方法可获得自然语言文本的特征（图 1.17）。

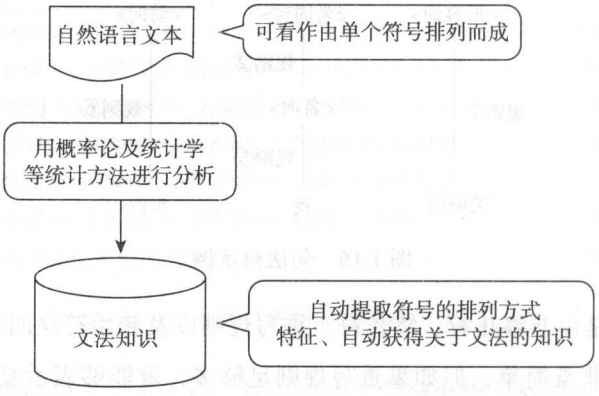


图 1.17 统计自然语言处理

统计自然语言处理是从 20 世纪 90 年代左右开始发展起来的。和统计自然语言处理的发展时期相一致的是互联网的爆发性发展，因而可以获得作为分析对象的庞大文本数据。统计自然语言处理的发展开拓了通往实用化、一般化语言处理技术的道路。

机器学习（machine learning）尤其是**深度学习**（deep learning）等方法也和统计方

法一起用到了自然语言处理领域，取得了非常大的成果。本章剩余部分将概括性地介绍什么是深度学习、如何将深度学习应用到自然语言处理中。

1.2 深度学习

本节将概述人工智能领域中的机器学习，并说明深度学习在其中所处的位置。

1.2.1 人工智能与机器学习

人工智能（Artificial Intelligence，AI）是模仿人类或生物的智能行为而形成有用软件的一门学科。人工智能包含多个不同的领域，而机器学习是其中一个重要支柱。

机器学习是指让机器或者计算机通过学习以变得越来越聪明的技术。图 1.18 给出了机器学习中各种各样的方法，其中深度学习是机器学习的一个分支。

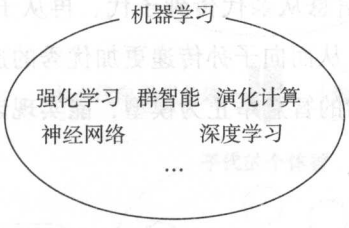


图 1.18 机器学习中各种各样的方法

在图 1.18 中，**强化学习**是基于对学习结果的评估而学习适当知识的机器学习方法（图 1.19）。例如，对于将棋[⊖]和围棋这样的游戏，当一局下完决出了胜负时，强化学习可以基于胜负结果学习到使下棋技能提升的对局知识。事实上，强化学习最初是从动物心理学出发而提出的一种学习理论。

⊖ 日本流行的一种棋类游戏。——译者注

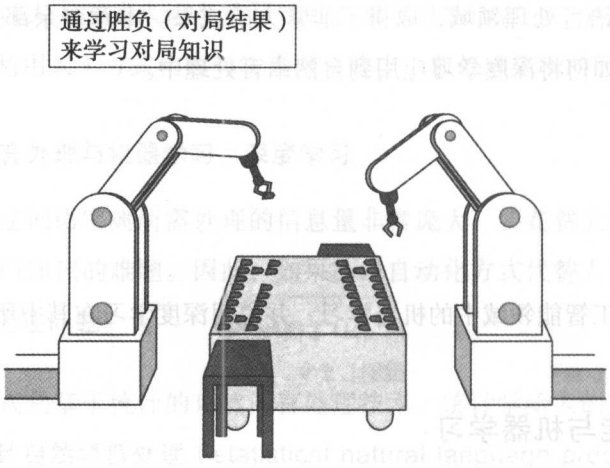


图 1.19 强化学习

图 1.18 中的**群智能**（swarm intelligence）及**演化计算**（evolutionary computation）是受生物种群和生物进化的启发而提出的学习方法。在鱼和鸟等群体中，作为群体构成要素的生物个体只执行单纯的动作行为，而生物群体作为一个整体能够具有捕食、回避障碍等智慧行为。群智能方法是受这种群体智慧行为的启发而提出的（图 1.20）。在生物进化过程中，生物的遗传信息从亲代传到子代、再从子代传到孙代时，通过和环境的相互作用进行淘汰和选择，从而向子孙传递更加优秀的遗传信息，这一生物进化机制启发了演化计算方法。以生物的智慧举止为模型，能实现诸如学习和最优化等智能行为（图 1.21）。

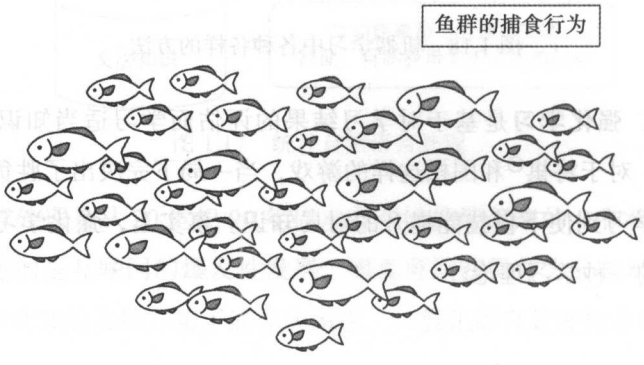


图 1.20 群智能：模型化生物的群体智慧行为

图 1.20 （续）

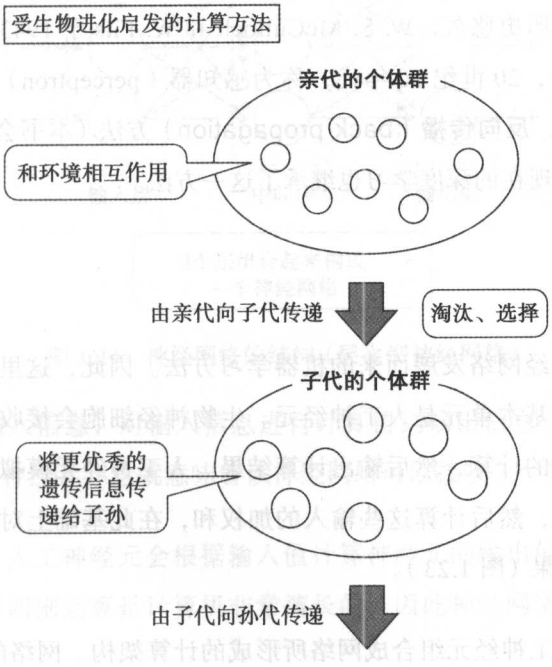


图 1.21 演化计算

同样，图 1.18 中的**神经网络**（neural network）也是一种模仿生物的计算方法。神经网络（也称为神经回路网络）本意是指由生物神经细胞构成的信息传递机构。受此启

发，可对生物神经细胞通过数学方法建立人工神经细胞模型——**人工神经元**（artificial neuron），这些人工神经元相互之间连接起来构成的网络就称为**人工神经网络**（artificial neural network）（图 1.22）。在人工智能和机器学习领域，人工神经网络一般简称为神经网络，下文也采用这一称呼。

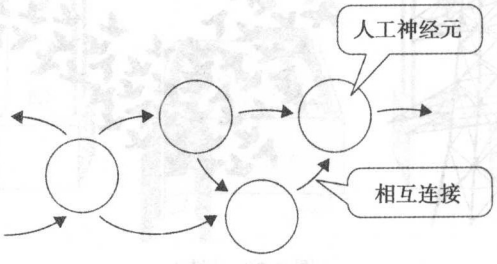


图 1.22 人工神经网络

神经网络的研究历史悠久，W. S. McCulloch 和 W. Pits 于 1943 年就提出了最基础的神经网络模型。此后，20 世纪 50 年代，称为感知器（perceptron）的神经网络被广为研究。20 世纪 80 年代，**反向传播**（back propagation）方法（本书会加以介绍）被广泛应用于各种不同领域，现在的深度学习也继承了这一方法。

1.2.2 神经网络

深度学习是从神经网络发展而来的机器学习方法。因此，这里对神经网络做一个介绍。构成神经网络的基本单元是人工神经元。生物神经细胞会接收其他多个神经细胞的输出信号并进行适当的计算，然后输出计算结果。人工神经元模拟生物神经细胞的处理机制，接收多个输入，然后计算这些输入的加权和，在此基础上对加权和的结果进行适当的计算后输出其结果（图 1.23）。

神经网络是将人工神经元组合成网络所形成的计算架构。网络的构成方法多种多样，例如，图 1.24 所示的是将多个人工神经元按层次进行配置的**层次型神经网络**的概念图。在图 1.24 中，每 3 个人工神经元组成一层，3 层组合起来，形成了由全体共 9 个人工神经元组成的神经网络。

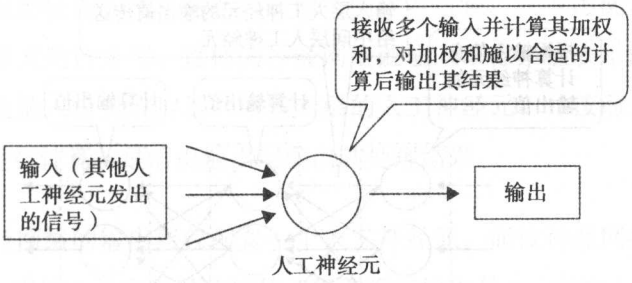


图 1.23　人工神经元的功能

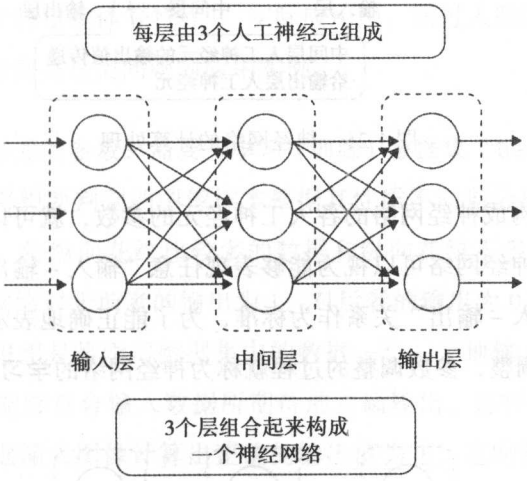


图 1.24　神经网络的结构（层次型神经网络）

神经网络接收输入信息，对输入信息进行计算得到输出信息。例如，在图 1.24 中，输入部分有 3 个人工神经元，这就意味着该神经网络可以接收 3 个独立的数值输入。

有了输入信息，人工神经元会根据输入值计算神经元的输出值。这里的计算大部分都是乘法或除法，而四则运算是计算机非常擅长的，因此神经网络能够高速地计算出输出信息。

人工神经元的输出值会传送给下一层（中间层）的人工神经元，在中间层上按同样方式进行计算。这样继续进行下去，在第三层（输出层）的人工神经元的输出，就是全体神经网络的输出（图 1.25）。

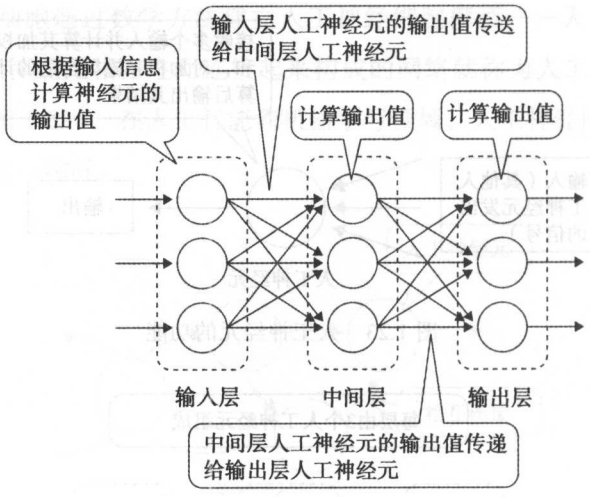


图 1.25 神经网络的计算处理

如果合适地设定构成神经网络的各人工神经元的参数，就可以自由地设定神经网络的整体行为。因此，神经网络可以视为能够表现任意“输入－输出”关系的万能信号转换器。这里给定“输入－输出”关系作为标准，为了能正确地表示这种关系，需要对人工神经元的参数进行调整，参数调整的过程就称为神经网络的**学习**⊖（图 1.26）。

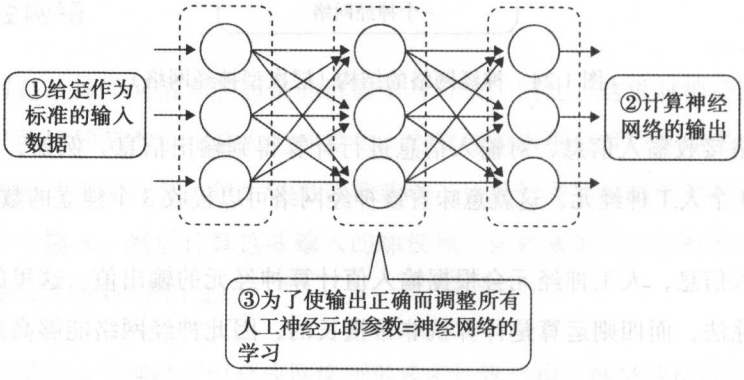


图 1.26 神经网络的学习

⊖ 神经网络的学习也可称为对神经网络进行训练，本书有时会将原文中的“学习”翻译为“训练”。——译者注

下面以图像识别任务为例来阐述神经网络的学习过程。先考虑配置神经网络的人工神经元时以纵横方式进行排列，这样的神经网络能输入以像素值表示的图像数据；再将图像的各个像素值输入给构成神经网络输入层的人工神经元，并按前述处理方法对各层人工神经元依次进行计算；最后由输出层输出其处理结果。

此处，在神经网络的输出层仅配置一个人工神经元，而该神经网络进行学习的最终目标是为了满足：当输入具有某种特性的一组图像时输出是 1，而输入具有其他特性的一组图像时输出是 0（这里的某种特性，可能是纵方向的花纹比较多、图像中含有某种特定图形、图像中具有某个特定物体等特性）。实际应用中，通过人脸图像进行身份认证、根据图像进行产品检查等都是这类问题的应用实例。

为了调整人工神经元的参数，需要有**学习（训练）数据集（training data set）**。上述图像判别的例子中，将想要判别的图像样本根据其性质不同划分成相应的图像数据，即为学习数据集。例如，将纵向花纹比较多的数据和纵向花纹不多的数据分别收集起来，训练神经网络，使得网络对于前者的输出为 1，对后者的输出为 0。具体的学习算法以后会加以说明，其基本思想是将学习数据集中的数据一个一个地输入神经网络中，调整网络参数，使网络输出能够符合输入数据所期待的正确输出。这样的调整会迭代很多次，直到神经网络能够根据输入图像计算出正确的输出值为止，这时神经网络的学习就结束了（图 1.27）。

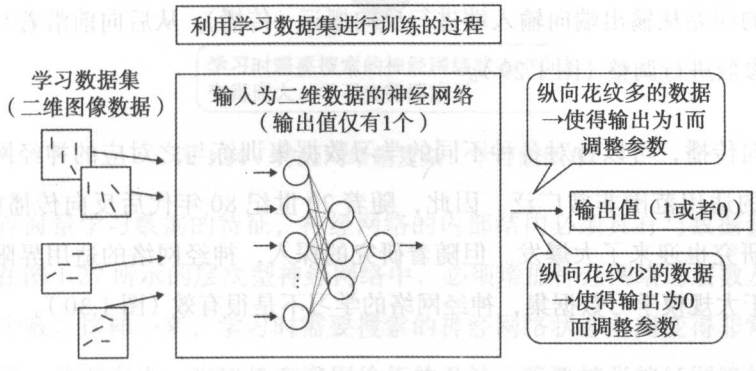


图 1.27　图像判别（1）

学习结束后，对于学习数据集中所包含的数据，神经网络的输出一定是正确的值。下面给神经网络输入学习时没有使用过的数据来测试神经网络的学习效果，这样的未知数据集称为**测试数据集（test data set）**（图 1.28）。

此时，如果神经网络对测试数据集也能够正确判别，就可以说神经网络的学习在**泛化**上取得了很好的效果。这里的泛化是指，将学习数据集中含有的共有特征提取出来，对于未包含在学习数据集中的数据也能得到和期望相符合的输出结果。

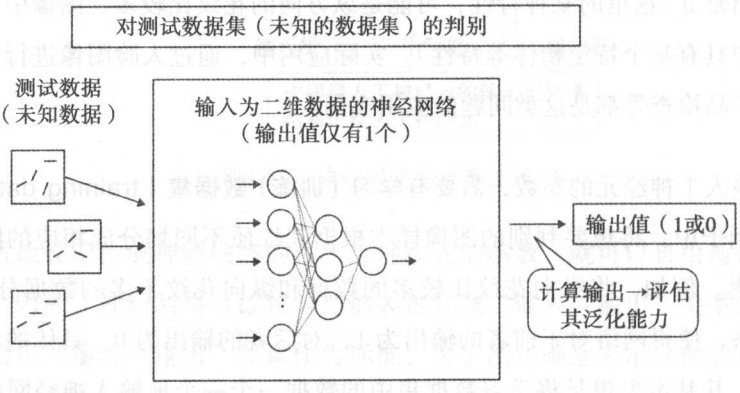


图 1.28 图像判别（2）

对于神经网络的学习，需要设计调整人工神经元参数的算法。为此，人们考虑了各种各样的方法，其中最标准的方法是前面提过的反向传播算法。反向传播是指利用神经网络的输出值中所含有的误差值来对各个人工神经元的参数进行微调整。其过程是将输出值所含有的误差从输出端向输入端进行反向搬运（传播），从后向前沿着与信号传递相反的方向对参数进行调整（图 1.29）。

使用反向传播，可以针对各种不同的学习数据集训练与之对应的神经网络，这就使得神经网络的适用范围非常广泛。因此，随着 20 世纪 80 年代后反向传播的广泛应用，神经网络的研究也迎来了大爆发。但随着研究的深入，神经网络的适用界限也变得非常明确，即对于大规模学习数据集，神经网络的学习不是很有效（图 1.30）。

20 世纪 90 年代以后，随着互联网的爆发式发展，研究者可以很容易地得到海量数据，于是他们开始尝试将海量数据作为学习数据而构建具有实用价值的系统。

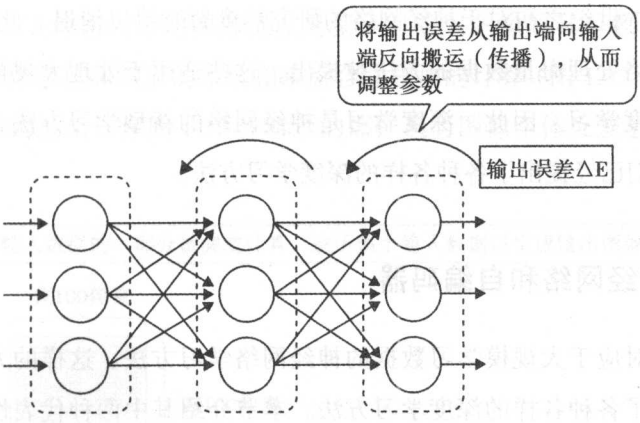


图 1.29　基于反向传播调整参数

和处理简单问题不同，将神经网络应用于海量数据时会产生本质上不同的问题。此时，为了在神经网络中保存海量数据的特征，需要构造复杂的神经网络结构。

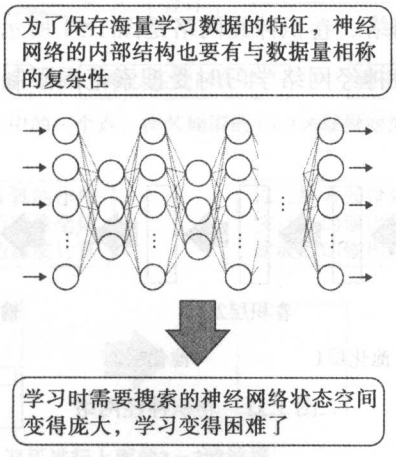


图 1.30　神经网络越复杂，学习越困难

为了保存海量学习数据的特征，神经网络的内部结构必须具有与数据量相称的复杂性。因此，在图 1.27 所示的层次型神经网络中，必须增加神经网络的层数及每层所含人工神经元的个数。这样一来，学习时需要搜索的神经网络状态空间变得非常庞大，学习就变得困难了。总而言之，构建具有实用价值的系统，需要扩张神经网络的结构使其与数据相称，这时神经网络的学习变得更加困难。

这样一来，21 世纪之初对于神经网络的研究热度暂时得以消退。此后，2010 年左右，多个通过神经网络处理海量数据的方法被提出，这些适用于处理大规模问题的神经网络学习方法就是深度学习。因此，深度学习是神经网络的新型学习方法。深度学习并不仅有一种，目前人们已经提出了各种各样的深度学习方法。

1.2.3 卷积神经网络和自编码器

深度学习是对应于大规模学习数据的神经网络学习方法。这样的方法不止一种，目前人们已经提出了各种各样的深度学习方法。本节介绍其中两种代表性方法：卷积神经网络和自编码器。

1. 卷积神经网络

深度学习中常用的一种方法是用**卷积神经网络**（Convolutional Neural Network，CNN）来构建大规模神经网络。卷积神经网络是图 1.31 所示的特殊形式的神经网络。采取这样的形式可以克服一般神经网络学习时受搜索范围限制的缺陷，从而能够学习大规模数据。

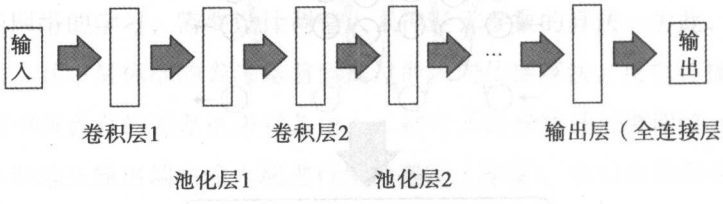


图 1.31 卷积神经网络

卷积神经网络的卷积层是针对输入进行卷积计算的人工神经元层。卷积计算是指将输入数据的一部分取出来进行适当的计算，且这样的计算处理在输入数据全体上遍历进行。

考虑针对二维图像采用二维滤波器，上述处理能更方便地处理图像数据。如图 1.32 ①所示，给定作为像素集合的二维图像，例如，100 × 100 的图像中共计有 10 000 个像素。假设对该图像采用 3 × 3 像素大小的图像滤波。

这里所说的 3 × 3 像素大小的图像滤波是指，对输入图像的某个像素周围 3 × 3 像素的范围中，将各个像素值和相应的系数相乘后合并相加的计算（图 1.32 ②）。合并相加的结果作为该滤波器的输出。像图 1.32 ③所示的那样遍历图像全体反复进行这一计算，就是卷积计算。

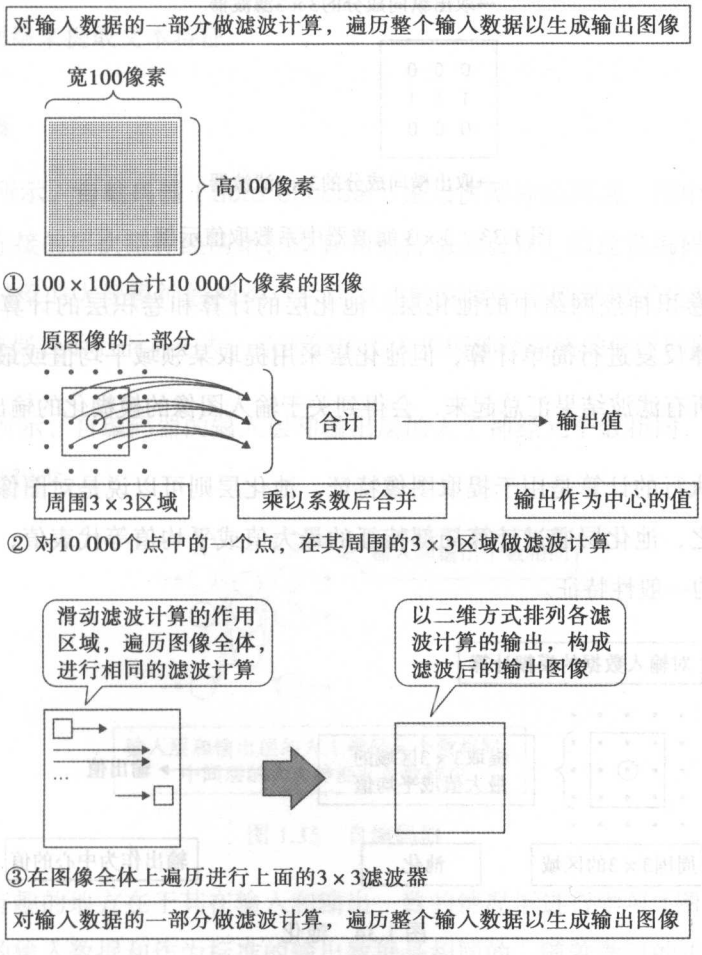


图 1.32　卷积计算

通过卷积计算可以根据滤波器系数的模式来提取原图像的特征（图 1.33 ）。例如，如果给滤波器纵向的系数配置较大的值，像素周边纵向的成分就会被提取出来；同样，如果给横向的系数配置较大的值，像素周边横向的成分就会被提取出来。在图像整体上遍历进行这样的计算，就是卷积计算。也就是说，卷积计算是指在输入数据整体上遍历进

行同样的计算处理，强化输入数据中某种特征的计算。

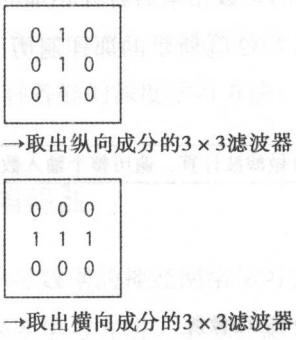


图 1.33　3×3 滤波器中系数取值示例

下面说明卷积神经网络中的池化层。池化层的计算和卷积层的计算相同，也是遍历输入数据整体反复进行简单计算，但池化层采用提取某领域平均值或最大值的滤波器（图 1.34）。将所有滤波结果汇总起来，会得到关于输入图像的模糊化的输出图像。

卷积层所执行的计算是用于提取图像特征，池化层则可以说是对图像做平均化模糊操作。总而言之，池化层通过计算局部特征的最大值或平均值等代表值，提取出和输入数据不太一样的一般性特征。

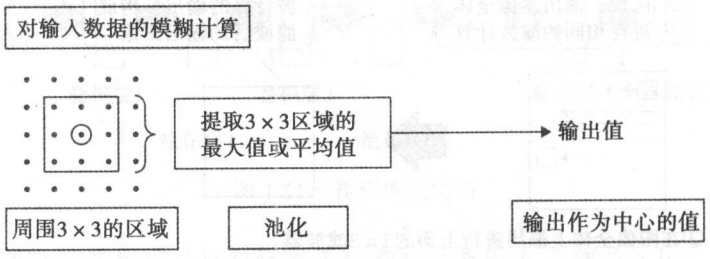


图 1.34　池化

卷积神经网络在多个不同层次中反复进行卷积和池化处理，这样的多层神经网络适用于大规模训练数据的学习。

此外，即使从学习这方面而言，卷积神经网络也比一般神经网络更为有利。因为用于卷积的滤波器尺寸很小，使得层间的连接比较简单，而各层内滤波器的参数可以共用，

因此对学习对象的解的搜索领域就非常小。这些特点使得卷积神经网络的学习比一般神经网络更为容易。

卷积神经网络在图像判别问题上取得了非常大的成功。此后，除了在图像识别领域，卷积神经网络在其他多个不同领域也取得了成功。例如，在自然语言处理领域，可尝试利用卷积神经网络来提取文本特征。

2. 自编码器

如图 1.35 所示，**自编码器**（auto encoder）是层次型神经网络。图中的神经网络是层间神经元全连接的层次型神经网络，没有其他特别的设计。因此自编码器自身并不是以深度学习为对象的大规模神经网络。后面叙述的深度学习则利用了自编码器的思想，在学习方法和数据表示上下了工夫，从而解决了大规模神经网络的学习问题。

如图 1.35 所示，自编码器的输入层和输出层的人工神经元个数相同，中间层的人工神经元个数相对较少。

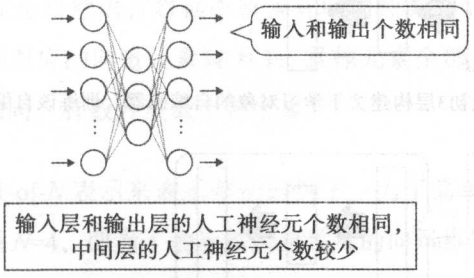


图 1.35　自编码器

自编码器有趣的地方在于其在输入和输出一致的情况下进行学习（图 1.36）。此时，构成训练数据的输入数据和作为标准的输出数据是相同的，随着学习的进行，最终能够得到输入和输出相一致的神经网络。

输入和输出相一致意味着经过神经网络的计算将输入数据原封不动地进行输出，从其计算结果中什么也得不到。事实上，自编码器的目的并不是由输入数据来计算输出数据，而是要利用中间层的输出。这里，中间层将输入数据的信息进行压缩并表示出来。

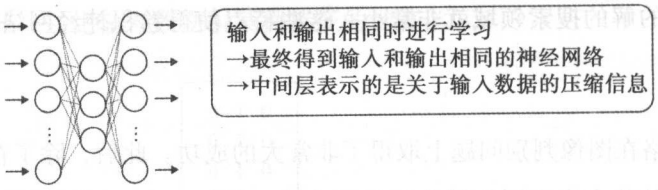
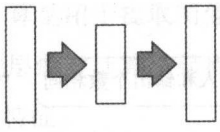


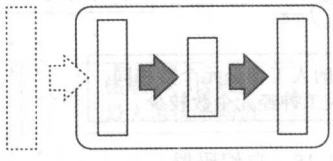
图 1.36　自编码器的学习

在自编码器中，中间层采用了比输入层个数更少的人工神经元。这样一来，如果输出数据能够重构出输入数据，就说明输入数据的特征在中间层中得到了良好的概括。换句话说，中间层输出的计算结果既保持了输入数据的特征，又实现了数据压缩。

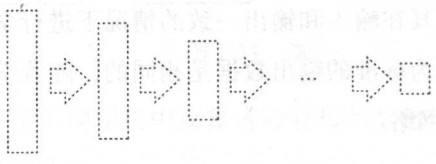
利用自编码器的这一特征可以提升多层神经网络的学习效率。其做法是，首先构造3层自编码器并训练该自编码器，然后去除输出层，将前一阶段的中间层作为输入层构成新的自编码器，再次训练这个新自编码器。重复这一过程，分阶段进行学习，最终就形成了多层神经网络（图 1.37）。



①由最初3层构建关于学习对象的自编码器，训练该自编码器



②将前一阶段的中间层作为输入层构成新自编码器，再次训练



③重复上述过程，构建多层神经网络

图 1.37　基于自编码器训练多层神经网络

也可将自编码器的思想应用于自然语言处理。例如，利用“自编码器能够对输入信

息进行概括”这一性质，在自然语言处理中就能够很好地解决如何表达单词意义的问题。下一节将介绍这方面的相关内容。

1.3 与自然语言处理相关的深度学习

本节将介绍如何将神经网络和深度学习方法应用于自然语言处理。

1.3.1 自然语言处理与神经网络、深度学习

为了将神经网络和深度学习等方法应用于自然语言处理，需要考虑如何将由自然语言所记录的非数值型数据转化为数值型数据。很显然，自然语言所记录的数据本身不是数值型的，但是，包括深度学习在内，神经网络的输入必须是数值型数据。因此，有必要设计将自然语言数据转化为数值型数据的方法。

一种方法是利用 1-of-N 表示来将单词表示为数值。1-of-N 表示是指利用 N 维向量来表示某个单词。这里的 N 是自然语言数据中所含有单词种类的个数。也就是说，要表示某个单词，只要将该单词对应的向量元素设为 1，其他元素全部设为 0。一般而言，N 的值和单词词典的单词数相同，有数万至数十万之多。

图 1.38 中给出了用 1-of-N 表示来表示单词的例子。为了简单起见，考虑单词种类只有 4 个的情形，也就是说 N=4，即为了表示单词所采用的向量维数是 4，该向量的元素个数是 4。单词则选用了图 1.12 所示的 4 个终止符“凛凛的”“美丽的”“我”以及“步行”。

这时，使向量的顶端元素对应于“凛凛的”，那么“凛凛的”这一单词用 1-of-N 表示就是（1,0,0,0）。同样，第二个元素对应于“美丽的”，则单词“美丽的”用 1-of-N 表示就是（0,1,0,0）。

如果能用 1-of-N 表示来将单词表示为数值，那么由连续单词形成的文本也可以通过 1-of-N 表示来表示出来。图 1.39 中将“我步行”及“美丽的我步行”等由多个单词组成的文本通过对单词向量的排列表示了出来。

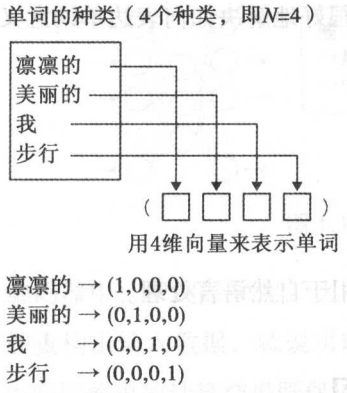


图 1.38 用 1-of-*N* 表示来表示单词

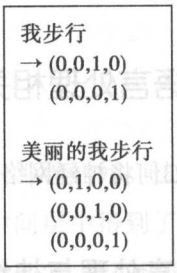


图 1.39 通过 1-of-*N* 表示来表示文本

1-of-*N* 表示是基础而精确的表示方法，但从信息检索和意义表达方面讲很难说它是一种好的表示方法。例如，考虑通过信息检索方法来搜索含有相似意义单词的文本。在检索由 1-of-*N* 表示的单词时，能检索出含有与检索词完全相同单词的文本，但不能检索出含有虽不完全相同但意义和用法相似单词的文本。而且，1-of-*N* 表示采用非常大的数据量来表示单词，其内容基本都是 0，这也是这种表示方法的不足之处。

因此，需要根据不同用途对 1-of-*N* 表示进行适当的改进。例如，在信息检索和单词的意义表达方面，人们提出了 bad-of-words 这一表示方法。这一方法将文本中所有单词的向量表示加起来合并成一个单一向量。如图 1.40 所示，将一个文本中所含单词的 1-of-*N* 表示集中起来，将这些向量的各个元素相加。相加的结果向量能表示该文本中哪个单词出现了几次，但是这样丢失了单词出现的顺序以及上下文联系等信息。该向量的各个元素表示的是文本中所出现单词的出现频率（Term Frequency，TF）。

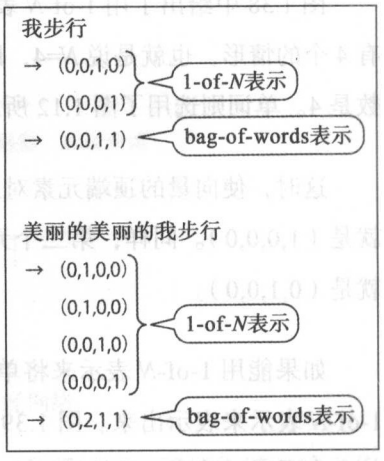


图 1.40 bag-of-words 表示

一般而言，考虑一个文本中所含意义只有一种的情形。这一来，可以认为相似文本中所含单词相互之间也具有相似的意义。因此，对于如何表达文本的意义以及如何综合表示相互之间意义相似的单词，bag-of-words 表示是一种有用的方法。

1.3.2 用神经网络来表达单词意义

在表达单词意义的方法中，神经网络也位列其中。与通过自编码器进行信息约简的思路相类似，这里采用的方法是利用神经网络的学习结果来表达单词的意义。

例如，考虑某含有 5 个连续单词的文本，假定单词相互之间所含意义相近。如图 1.41 所示，以第 n 个单词 n 为中心，将其前面两个单词和后面两个单词作为输入数据，将中心单词 n 作为输出，构建神经网络，并对该网络进行训练。当采用各种不同的数据进行学习后，神经网络参数中就储存了关于某单词在什么样的单词集合中出现等信息。这样存储起来的神经网络参数集合可以视为该单词的意义。这样的表示方法称为**连续 bag-of-words 表示**（Continuous Bag-Of-Words，CBOW）。

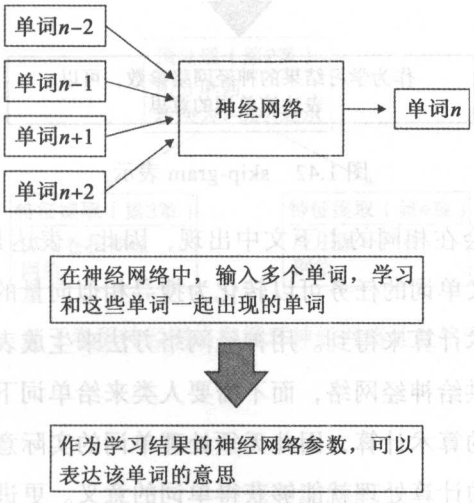


图 1.41 连续 bag-of-words 表示

此外，和连续 bag-of-words 相反，将某单词作为输入，将其前后的单词组合作为输出构成神经网络，其学习结果也可以表达该单词的意思。这样的表示方法称为 **skip-gram 表示**（图 1.42）。

连续 bag-of-words 表示和 skip-gram 表示都是用构成神经网络的人工神经元的参数集合来表达单词的意义。如果适当地设定神经网络的规模，则可以用比单词的种类数少很

多的参数个数来表达单词的意义。因此，对它们进行比较，1-of-*N* 表示需要含有非常多元素的高维向量，而连续 bag-of-words 表示和 skip-gram 表示能实现低维向量表示，这为深度学习改善学习效率提供了线索。

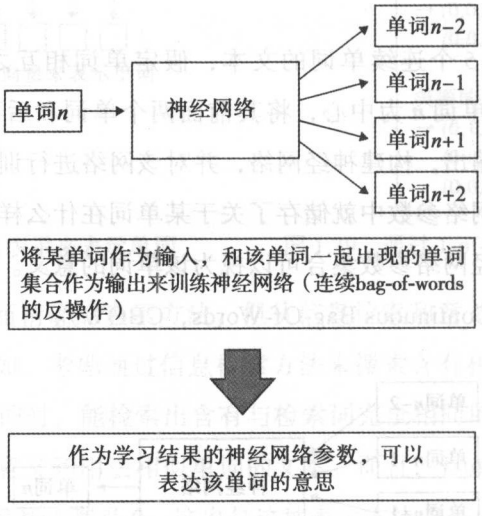


图 1.42 skip-gram 表示

意义相类似的单词会在相同的上下文中出现，因此，表达其意义的向量也相类似。这样一来，搜索相似意义单词的任务可以转化为搜寻相似向量的问题，而向量之间的相似度可以通过单纯的算术计算来得到。用神经网络方法来生成表达意义的向量时，只需将例文分解为单词后提供给神经网络，而不需要人类来给单词下一个定义。而且，相似度的计算也仅仅是单纯的算术计算，因此无须处理单词的实际意义。这就意味着，不需要人类的参与，纯机械的计算处理就能够获得单词的意义。更进一步，基于向量的意义表达、意义表达相互间的数值比较以及意义表达相互间的加减计算都成为可能，这是以往的意义表达方法所未能考虑过的神经网络方法的特征。

连续 bag-of-words 表示和 skip-gram 表示可以通过使用 word2vec [⊖] 等工具来进行尝试。

⊖ 关于 word2vec，可以参考网页（2017 年 2 月）https://code.google.com/archive/p/word2vec/。

1.3.3 深度学习应用于自然语言处理

利用 1-of-*N* 表示等方法将自然语言转化为数值型数据后，如何才能较好地将深度学习应用到这些数值处理上呢？本书其余章节将按顺序介绍其方法。

首先，作为应用深度学习方法的前提，第 2 章将分析对日语的处理方法及单词的切分处理等相关话题。第 3 章通过示例来介绍如何应用卷积神经网络提取文本的特征。随后，第 4 章介绍能利用层次型神经网络处理单词出现顺序的循环神经网络（Recurrent Neural Network，RNN），以及将循环神经网络应用于自然语言处理的方法。通过上述分析，探讨自然语言处理和深度学习之间的关系（图 1.43）。

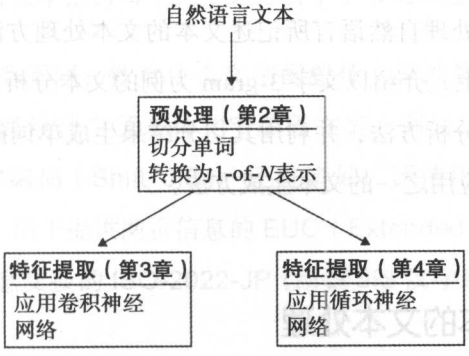


图 1.43 基于卷积神经网络和循环神经网络的自然语言处理

CHAPTER 2

第 2 章

基于文本处理的自然语言处理

本章讲述由程序来处理自然语言所记述文本的文本处理方法。具体来说，在以文字为单位进行处理的基础上，介绍以文字 3-gram 为例的文本分析方法；然后，讲述对单词进行切分处理的形态素分析方法，并利用其切分结果生成单词的 1-of-*N* 表示。此外，还将述及作为这些技术的应用之一的文本生成方法。

2.1 自然语言文本的文本处理

本节说明如何将作为输入的自然语言文本转换为数值表示的方法。数值表示是深度学习方法的处理对象。这里假定作为输入的自然语言是在 Windows 环境下采用 Shift_ JIS 汉字编码来表示的。

2.1.1 文字处理

首先看看如何以文字为单位进行处理，它是以单词为单位的处理及基于 1-of-*N* 表示的处理等方法的基础。

1. 自然语言文本的表示

自然语言文本是通过文字的排列表示出来的。这里的文字是用**文字编码**来表示的，

而文字编码是为表示某个文字所使用的二进制符号。

文字编码是二进制数，其必要的取值范围根据所需要表示的文字种类数而定。例如，如果作为表示对象的文字仅仅是字母、数字和一些代表性符号，则需要表示的文字种类数不会超过几百个。这时，文字编码二进制数的取值范围从 0 到 255（即 2^8-1）就非常充分了。也就是说，英文字母、数字、符号等文字的表示一般可通过 8 位二进制数（即 1 字节）的文字编码来实现。

和英文字母、数字、符号等不同，日语中使用的平假名、片假名、汉字等的文字种类数已经增长到几万种，因此，为了表示日语文字，每个文字需要 2 字节或更多字节来表示，也就是说，日语的自然语言文本是由每个文字通过若干字节的二进制数表示而集合起来的。

随着文字编码的种类不同，针对什么文字配置什么样的数值也各不相同。人们根据用途和目的的不同已经设计出了多种文字编码方案，例如，Windows 下的日语文本文件采用 **Shift_JIS 汉字编码（Shift_JIS）**。还有诸如广泛使用的用于程序内部处理的 Unicode 文字编码体系、用于提供网页信息的 **EUC（Extended UNIX Code）**、用于电子邮件等互联网上进行信息交换的 **ISO-2022-JP（所谓 JIS 汉字编码）**标准等，都是文字编码的例子。

本节将 Windows 下文本文件中存放的日语文本作为处理对象，其文字编码限定为 Shift_JIS 汉字编码。通过 Shift_JIS 汉字编码表示的日语文本的结构如图 2.1 所示。

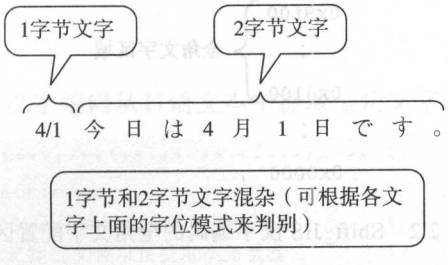


图 2.1 通过 Shift_JIS 汉字编码表示日语文本

在 Shift_JIS 汉字编码中，1 个文字可以通过 1 字节或 2 字节来表示。主要表示方式为：1 字节表示的文字是字母、数字以及符号等，2 字节表示的文字是汉字以及其他

一些符号。通常，1字节表示的文字称为**半角文字**，2字节表示的文字称为**全角文字**。基于Shift_JIS汉字编码的日语文本中，一般会混杂着半角文字和全角文字。以日语为对象进行自然语言处理时，既需要从作为分析对象的文本中提取表示数字和符号等的半角文字，也需要提取表示汉字、平假名、片假名的全角文字。下面介绍文字的提取方法。

检查由1字节的二进制数所表示的编码值可以发现，Shift_JIS汉字编码中附带了能够区别这一编码值是表示半角文字还是表示全角文字第一个字节的方法。这一区别方法可通过如下方式来判断。其中以0x开头的数值是以十六进制数来表示的。

> 如果某1字节的数值是0x81以上0x9f以下或者0xe0以上0xef以下，则这一数值表示的是全角文字的第一个字节。如果在这个范围以外，则这一数值表示的是全角文字以外的文字。

之所以能够通过这一方法做出判定，是因为Shift_JIS汉字编码自身所采用的编排方式。如图2.2所示，全角文字被固定配置在特定范围内。

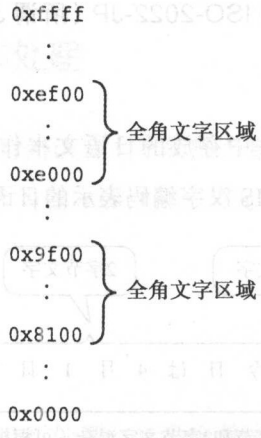


图2.2 Shift_JIS汉字编码的全角文字配置区域

利用这一性质就能从给定文本中提取出作为分析对象的全角日语文字部分。例如，当给出图2.3所示的文章时，其中包含了作为分析对象的全角日语文字、半角字母、半角数字及换行符号等。利用图2.2的性质，考虑设计出提取全角文字的程序。

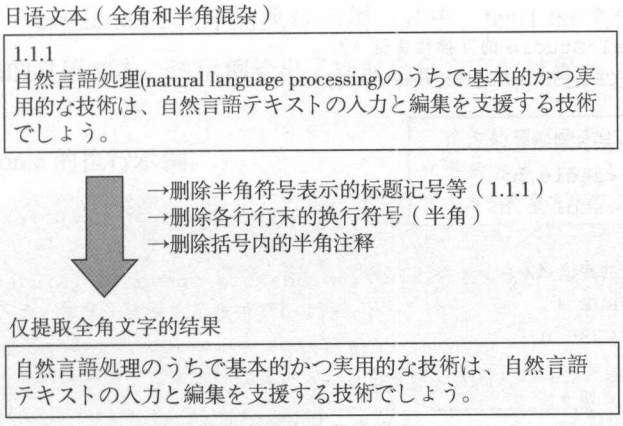


图 2.3 根据 Shift_JIS 汉字编码提取日语文本中的全角文字

基于 C 语言程序来实现提取全角文字的全过程如下。

提取全角文字的过程（extraction.c）

重复以下操作直到文件输入终止

从输入文件中读入 1 字节的数据

如果输入数据是全角文字的第一个字节，将其和下一个字节的数据一起输出

清单 2.1 给出了用 C 语言来实现这一处理的过程。extraction.c 程序由 main() 函数和 is2byte() 函数组成。其中 is2byte() 函数用于判断所给的参数是不是全角文字的第一个字节（0x81 以上 0x9f 以下，或者 0xe0 以上 0xef 以下），其结果用符号常数 TRUE 或 FALSE 返回。

清单 2.1 基于 Shift_JIS 汉字编码从日语文本中提取全角文字的 extraction.c 程序。

```
/*******************************************/
/*          extraction.c                   */
/*  用于 Shift JIS 汉字编码的全角文字提取器        */
/*  从 Shift_JIS 记述的文件中仅提取全角数据         */

/* 使用方法                                  */
/*C:\Users\odaka\ch2>extraction < text1.txt  */
/*******************************************/
```

```
 9
10 /* 和 Visual Studio 的互换性保证 */
11 #define _CRT_SECURE_NO_WARNINGS
12
13 /*include 头文件 */
14 #include <stdio.h>
15 #include <stdlib.h>
16
17 /* 符号常数的定义 */
18 #define TRUE 1
19 #define FALSE 0
20
21 /* 函数原型声明 */
22 int is2byte(int chr); /* 判断是否是全角文字的第一个字节 */
23
24 /*******************************************/
25 /*            main() 函数                  */
26 /*******************************************/
27 int main()
28 {
29 int chr;/* 输入文字 */
30
31 /* 读入数据后以一个文字接一个文字的方式输出 */
32 while ((chr = getchar()) != EOF){
33  if (is2byte(chr) == TRUE){
34    /* 输出全角 (2 字节) */
35    putchar(chr);
36    putchar(getchar());
37  }
38 }
39
40 return 0;
41 }
42
43 /*******************************************/
44 /*  is2byte() 函数                         */
45 /* 判断是否是全角文字的第一个字节          */
46 /*******************************************/
47 int is2byte(int c)
48 {
49   if (((c >= 0x81) && (c <= 0x9F)) || (c >= 0xe0) && (c <= 0xef))
50   return TRUE; /*2 字节文字 */
51 return FALSE; /*1 字节文字 */
52 }
```

实例 2.1 给出了 extraction.c 程序的执行示例。其中，text1.txt 文件中存放的是基于 Shift_JIS 汉字编码的日语文本，该示例给出了提取全角文字的过程。

实例 2.1 extraction.c 的执行示例。

作为处理对象的 text1.txt 文件的内容（text1.txt 文件中全角和半角文字混杂在一起）

```
c:\Users\odaka\ch2>type text1.txt
1.1.1
自然言語処理 (natural language processing) のうちで基本的かつ実用的な技術は、自然言語テキストの入力と編集を支援する技術でしょう。
c:\Users\odaka\ch2>extraction < text1.txt
自然言語処理のうちで基本的かつ実用的な技術は、自然言語テキストの入力と編集を支援する技術でしょう。
c:\Users\odaka\ch2>
```

执行 extraction.c 程序（仅提取全角文字）

对 extraction.c 稍做修改，就能够生成全角文字的 *n*-gram 表示。首先，考虑生成全角文字的 1-gram 方法（图 2.4）。

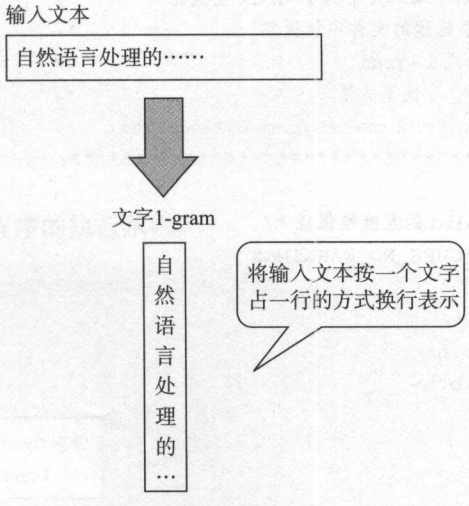


图 2.4 生成文字 1-gram 表示

如图 2.4 所示，文字 1-gram 是将输入文本按照一个文字占一行的方式换行表示的。将 extraction.c 程序的处理按如下方式修改后，就能从全角文字和半角文字混杂在一起的文本中得到关于全角文字的 1-gram 表示。为获得 1-gram 表示所做的修改非常少，仅在输出部分增加对每一文字的换行处理即可。

生成文字 1-gram 表示的过程（make1gram.c）

重复以下操作直到文件输入终止

从输入文件中读取 1 字节数据

如果输入数据是全角文字的第一个字节，执行下面的操作

输出该字节和其后的一个字节数据

输出换行符（1-gram 的切分）

清单 2.2 给出了用 C 语言来实现上述处理的程序，实例 2.2 给出了 make1gram.c 程序的执行示例。

清单 2.2 生成文字 1-gram 表示的 make1gram.c 程序。

```
/*********************************************/
/*          make1gram.c                      */
/*  Shift_JIS 汉字编码文字的 1-gram 生成器   */
/*  从 Shift_JIS 记述的文件中仅提取          */
/*   全角数据并生成 1-gram                   */
/*                使用方法                   */
/*C:\Users\odaka\ch2>make1gram < text1.txt   */
/*********************************************/

/* 和 Visual Studio 的互换性保证 */
#define _CRT_SECURE_NO_WARNINGS

/*include 头文件 */
#include <stdio.h>
#include <stdlib.h>

/* 符号常数的定义 */
#define TRUE 1
#define FALSE 0

/* 函数原型的声明 */
int is2byte(int chr); /* 判断是否是全角文字的第一个字节 */

/*********************************************/
/*            main() 函数                    */
/*********************************************/
```

```
int main()
{
 int chr;/* 输入文字 */

/* 读入数据后以一个文字接一个文字的方式输出 */
while ((chr = getchar()) != EOF){
  if (is2byte(chr) == TRUE){          /* 如果是全角文字 */
    putchar(chr);
 /* 输出全角文字的第二个字节 */
 putchar(getchar());
 putchar('\n');                       /*1-gram 的切分 */
}
}

 return 0;
}

/*********************************************/
/*    is2byte() 函数                          */
/* 判断是否是全角文字的第一个字节              */
/*********************************************/
int is2byte(int c)
{
if (((c >= 0x81) && (c <= 0x9F)) || (c >= 0xe0) && (c <= 0xef))
 return TRUE; /*2 字节文字 */
return FALSE; /*1 字节文字 */
}
```

实例 2.2 make1gram.c 程序的执行示例。

```
c:\Users\odaka\ch2>make1gram < text1.txt
自
然
言
語
処
理
のうちで
```

将全角文字一个一个地切分出来（生成 1-gram）

```
(下面继续输出)
```

2. 文字 3-gram 分析

下面考虑编写生成文字 3-gram 表示的程序，其算法如下所示，非常简单。

生成文字 3-gram 表示的过程（make3gram.c）

重复以下操作直到文件输入终止

从输入文件中读取 1 字节的数据

如果输入数据是全角文字的第一个字节，执行如下操作

将读取的全角文字的第一个字节传给 put3gram() 函数

读取下一字节的数据，传给 put3gram() 函数

其中，子程序 put3gram() 函数的处理过程如下所示。

put3gram() 函数的处理

添加 3 个全角文字到队列参数中

如果文字是按 2 字节切分，输出队列中存放的 3 个文字

在上面的 put3gram() 函数中，函数内部保存着 3 个文字组成的队列。向队列中依次添加文字数据，当添加的是由 2 字节组成的 1 个文字数据时，输出队列中存放的 3 个文字数据，该输出就是文字 3-gram 表示（图 2.5）。

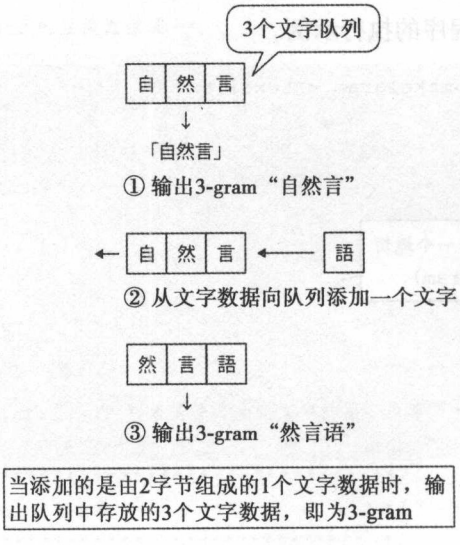


图 2.5 put3gram() 函数的处理

清单 2.3 给出了根据上述考虑所编写的 make3gram.c 程序。

清单 2.3 make3gram.c 程序的源代码。

```
/*********************************************/
/*            make3gram.c                    */
/*  Shift_JIS 汉字编码文字的 3-gram 生成器    */
/*      从 Shift_JIS 记述的文件中仅提取       */
/*      全角数据并生成 3-gram                 */
/* 使用方法                                   */
/*C:\Users\odaka\ch2>make3gram <text1.txt     */
/*********************************************/

/* 和 Visual Studio 的互换性保证 */
#define _CRT_SECURE_NO_WARNINGS

/*include 头文件 */
#include <stdio.h>
#include <stdlib.h>

/* 符号常数的定义 */
#define TRUE 1
#define FALSE 0
#define N 6 /*n-gram 的 n 的 2 倍 */

/* 函数原型的声明 */
int is2byte(int chr) ;          /* 判断是否是全角文字的第一个字节 */
void put3gram(int chr);         /* 输出 3-gram*/
int invert(int flag) ;          /* 反转 flag*/

/****************/
/*  main() 函数  */
/****************/
int main()
{
int chr ;/* 输入文字 */

/* 读入数据后以一个文字接一个文字的方式输出 */
while((chr=getchar())!=EOF){
if(is2byte(chr)==TRUE){
/* 根据 put3gram() 函数输出 */
put3gram(chr) ;
put3gram(getchar()) ;
}
```

```
41 }
42
43 return 0 ;
44 }
45
46 /*******************************/
47 /*    invett() 函数              */
48 /*   反转 flag                   */
49 /*******************************/
50 int invert(int flag)
51 {
52 if(flag==FALSE)
53  return TRUE ;
54 return FALSE ;
55 }
56
57 /*******************************/
58 /*  put3gram() 函数              */
59 /*  输出 3-gram                  */
60 /*******************************/
61 void put3gram(int c)
62 {
63 static char queue[N]="    " ;          /* 用于输出的队列 */
64 static int flag=FALSE ;                /* 关于输出的时间控制 */
65 int i;                                 /* 循环控制 */
66
67 /* 向队列中添加数据 */
68 for(i=0;i<N-1;++i)
69  queue[i]=queue[i+1] ;
70 queue[N-1]=c ;                         /* 添加数据 */
71
72 /* 如果是 2 字节切分，就输出 */
73 if(flag==TRUE){
74  for(i=0;i<N;++i)
75   putchar(queue[i]) ;
76   putchar('\n') ;
77  }
78  /* 反转 flag*/
79  flag=invert(flag) ;
80 }
81
82 /*******************************/
83 /*    is2byte() 函数             */
84 /* 判断是否是全角文字的第一个字节    */
```

```
/*******************************/
int is2byte(int c)
{
if(((c>0x80)&&(c<0xA0))||(c>0xDF)&&(c<0xF0))
 return TRUE ; /*2 字节文字 */
return FALSE ; /*1 字节文字 */
}
```

实例 2.3 给出了 make3gram.c 程序的执行示例。

实例 2.3 make3gram.c 程序的执行示例。

```
c:\Users\odaka\ch2>make3gram < text1.txt
    自
   自然
  自然言
  然言語
  言語処
  語処理
  処理の
  理のう
  のうち
  うちで
  ちで基
(下面继续输出)
```

当添加 1 文字数据时，输出队列中存放的 3 个文字数据作为 3-gram

如第 1 章所示，使用 3-gram 可以对文本进行分析。为此，需要对 3-gram 进行分类，统计其出现频率。这一处理可以通过如下过程实现。

① 生成 3-gram（make3gram.c 程序）

↓

② 按词典顺序对 3-gram 进行整理

↓

③ 将相同的 3-gram 在一行中汇合，将其重复次数置于行首处

↓

④ 将行首的重复次数按降序排列

在上述过程中，步骤②是将文本中出现的 3-gram 按一定的规则进行整理。其结果如

例 2.4 所示，多次出现的 3-gram 会被集中到同一地方，而相同的 3-gram 会重复出现好多行。可以使用 Windows 系统中标准配置的 sort 命令来实现这一处理。

实例 2.4 3-gram 出现频率分析（1）。

```
c:\Users\odaka\ch2>type text2.txt
すももももももものうち
c:\Users\odaka\ch2>make3gram < text2.txt
   す
  すも
すもも
もも
もも
もも
もも
もも
もも
もも
もも
もの
ものう
のうち
c:\Users\odaka\ch2>make3gram < text2.txt | sort
す
  すも
すもも
のうち
ものう
もの
もも
もも
もも
もも
もも
もも
c:\Users\odaka\ch2>
```

在作为处理对象的 text2.txt 中，存放有“すももももももものうち”文本

使用 make3gram 生成 3-gram

使用 sort 命令对 3-gram 排序

在步骤③中，相同的 3-gram 被汇总到一行中，其重复次数被添加到行首。由此，就能够求出各个 3-gram 的出现次数（实例 2.5）。在 UNIX 中，使用标准的 uniq 命令（option c）能够实现这一处理。在 Windows 环境下，没有对应的命令，本书将和 UNIX 下的 uniq 命令（option c）相同的执行过程编写成 uniqc.c 程序，其源代码在附录 A 中给出。

实例 2.5 3-gram 出现频率分析（2）。

```
c:\Users\odaka\ch2>make3gram < text2.txt | sort | uniqc
1          す
1         すも
1        すもも
1       のうち
1       ものう
1       もの
1       もも
6       もも
c:\Users\odaka\ch2>
```

使用 uniqc.c 程序将同一 3-gram 的重复次数添加到行首

最后的步骤④按照行首重复次数的降序方式对③的结果进行整理。实例 2.6 给出了这一处理的示例。在该处理中，对行首的数值进行降序排列。UNIX 中的 sort 命令（option n）可以用于这一处理。在 Windows 环境下，执行这一处理的程序是 sortn.c，其源代码在附录 B 中给出。

实例 2.6 3-gram 出现频率分析（3）。

```
C:\Users\odaka\ch2>make3gram < text2.txt | sort | uniqc | sortn
6       もも
1       すも
1       のうち
1       ものう
1       もの
1         すも
1          す
C:\Users\odaka\ch2>
```

利用 sortn.c 程序对 3-gram 的出现频率进行降序排列

2.1.2 单词处理

前一节讲述了文字处理这一准备工作，下面对单词进行处理。首先介绍如何从文本中将单词切分出来，然后利用切分结果生成单词 2-gram。

1. 单词 1-gram 的获取

如第 1 章所述，在自然语言处理领域中应用更多的是以单词为基本单位进行处理的模型。因此，有必要探讨从文本中提取单词的方法。然而，和英语不同，日语单词之间不能通过空格来分开。下面考虑采用何种方法来将用日语记述的自然语言文本的基本单

位——单词切分出来。

一种简单的方法是将文本通过文字的种类加以切分。图 2.6 是这一方法的示例。图中字的种类有汉字、片假名、标点符号及其他文字，之所以将标点符号和其他文字分开处理，是因为标点符号肯定会被作为单词而切分。

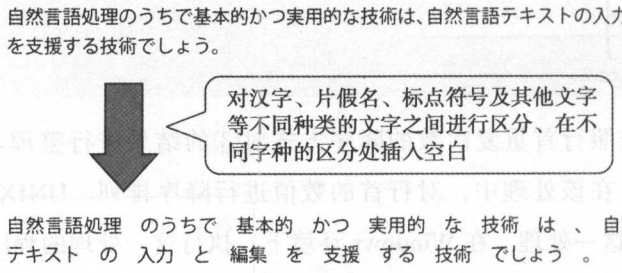


图 2.6 文本中通过区分文字种类的方法切分出单词的示例

在图 2.6 中，文本开头的“自然言語処理”这一连串单词组成的文字序列作为一个整体被切分出来。随后，由汉字、片假名之外的平假名这一文字种类组成的连续文字序列“のうちで”被切分为单词。此后，当文字种类发生变化时，就会切分出相应的单词。

通过程序来实现上述处理机制时，可以利用 Shift_JIS 汉字编码的编排特征。Shift_JIS 汉字编码中汉字和片假名配置见表 2.1，这一编排特征可用来区分文字种类。

表 2.1 Shift_JIS 汉字编码中汉字和片假名的配置

数值范围	文字种类
0x8800 以上	汉字
0x8340～0x8396	片假名

有了表 2.1 的配置信息，加上按其他方式对标点符号进行的处理，就可以给出如图 2.6 所示的单词切分的必要判别条件。具体来说，就是执行下面的处理方式。

根据文字种类切分单词的过程（makew1gram.c）

重复以下过程直到文件输入终止

从输入文件中读入 1 字节数据
  如果输入数据是全角文字的第一个字节，执行如下操作
    如果文字种类发生了变化，输出换行符号（切分单词）
    将该字节和下一个字节数据同时输出

在上述过程中，文字种类的判别过程如下所示。

文字种类的判别过程（type() 函数）
将全角文字的第一个字节设为 chr1，第二个字节设为 chr2
如果（chr1>=0x88），则为汉字
如果（chr1==0x83）&&（chr2>=0x40）&&（chr2<=0x96），则为片假名
如果全角文字是“、”“。”“，”“．”，则为标点符号
如果和上述情况都不一致，则为其他文字种类

清单 2.4 给出了按以上过程所实现的程序。

清单 2.4 makew1gram.c 程序。

```
/*************************************************/
/*            makew1gram.c                       */
/*  Shift_JIS 汉字编码文字的单词 1-gram 生成器       */
/*  从 Shift_JIS 记述的文件中仅提取全角              */
/*  数据并生成单词的 1-gram                         */
/* 使用方法                                        */
/*C:\Users\odaka\ch2>makew1gram < text1.txt        */
/*************************************************/

/* 和 Visual Studio 的互换性保证 */
#define _CRT_SECURE_NO_WARNINGS

/*include 头文件 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

```

```
/* 符号常数的定义 */
#define TRUE 1
#define FALSE 0
#define KANJI 0                    /* 文字种类为汉字 */
#define KATAKANA 1                 /* 文字种类为片假名 */
#define KUTOUTEN 2                 /* 文字种类为标点符号 */
#define SONOTA 3                   /* 文字种类为上述之外的类型 */

/* 函数原型的声明 */
int is2byte(int chr) ;             /* 判断是否是全角文字的第一个字节 */
int typep(int chr1,int chr2);/* 文字种类的判定 */

/****************/
/* main() 函数   */
/****************/
int main()
{
 int chr1,chr2 ;/* 输入文字 */
 int now,last=-1 ;/* 记忆字种 */

/* 读入数据后以一个文字接一个文字的方式输出 */
while((chr1=getchar())!=EOF){
  if(is2byte(chr1)==TRUE){
   chr2=getchar() ;
   now=typep(chr1,chr2) ;

   /* 输出全角（2字节）*/
   if(now!=last){
    putchar('\n') ;/* 切分 1-gram*/
    last=now ;
   }
   putchar(chr1) ;
   putchar(chr2) ;
  }
 }

 return 0 ;
}

/*******************************/
/*   typep() 函数              */
/*  文字种类的判定             */
/*******************************/
int typep(int chr1,int chr2)
```

```
62 {
63  char chr[3]="  " ;/* 标点符号判定用 */
64
65  chr[0]=chr1 ; chr[1]=chr2 ; /* 文字的设定 */
66  /* 文字种类的判定 */
67  if(chr1>=0x88)
68   return KANJI ;/* 汉字 */
69  else if((chr1==0x83)&&(chr2>=0x40)&&(chr2<=0x96))
70   return KATAKANA ;/* 片假名 */
71  else if((strncmp(chr,"。",2) ==0) ||
72          (strncmp(chr,". ",2) ==0) ||
73          (strncmp(chr,"、",2) ==0) ||
74          (strncmp(chr,", ",2) ==0))
75   return KUTOUTEN ;/* 标点符号 */
76  return SONOTA ;/* 其他 */
77 }
78
79 /*******************************/
80 /*  is2byte() 函数               */
81 /* 判断是否是全角文字的第一个字节    */
82 /*******************************/
83 int is2byte(int c)
84 {
85  if(((c>0x80)&&(c<0xA0))||(c>0xDF)&&(c<0xF0))
86   return TRUE ;/*2 字节文字 */
87  return FALSE ;/* 1 字节文字 */
88 }
```

实例 2.7 给出了清单 2.4 中 makew1gram.c 程序的执行示例。

实例 2.7 makew1gram.c 程序的执行示例（1）。

```
C:\Users\odaka\ch2>makew1gram < text1.txt
自然言語処理
のうちで
基本的
かつ
実用的
な
技術
は
、
自然言語
テキスト
```

通过切分单词生成单词 1-gram

```
の
入力
と
編集
を
支援
する
技術
でしょう
。
C:\Users\odaka\ch2>
```

在实例 2.7 的执行示例中，乍一看能看到很好的结果，但实际上切分出不合适单词的可能性是存在的。在实例 2.8 中，如果输入文本全部是同一文字种类形成的词链，就无法找到区分点，从而根本无法切分。事实上，makew1gram.c 程序只能执行非常简单的处理，这是其局限性。

实例 2.8 makew1gram.c 程序的执行示例（2）。

```
不适用的例子
C:\Users\odaka\ch2>type text2.txt
すももももももものうち
C:\Users\odaka\ch2>makew1gram < text2.txt

すももももももものうち
C:\Users\odaka\ch2>
```

同一文字种类形成的词链，无法找到区分点

为了切分出更为合适的单词，需要通过更深入的语言知识和词典进行分析。本书不深入探讨这些技术，通过使用形态素分析工具等方式也能够很简单地加以实现。

在实例 2.9 中，给出了使用 MeCab ⊖这一**形态素分析**工具的例子。这里通过 extraction.c 程序从 text1.txt 文件中提取出全角文字，将其结果通过 mecab 命令切分出形态素。使用 MeCab 工具，不仅能够切分出单词，还能给出各个单词的读法和语法作用、单词的原型等信息。当然，如果在命令中使用 -O wataki 选项，也可以只执行单词的切分任务。

⊖ MeCab 是日本京都大学信息学研究科和日本电信电话株式会社交流科学基础研究所开发的工具，从 http://taku910.github.io/mecab/ 可以下载（2017 年 2 月）。

实例 2.9 使用 MeCab（形态素分析工具）的例子。

```
C:\Users\odaka\ch2>extraction < text1.txt | mecab
自然    名詞,形容動詞語幹,*,*,*,*,自然,シゼン,シゼン
言語    名詞,一般,*,*,*,*,言語,ゲンゴ,ゲンゴ
処理    名詞,サ変接続,*,*,*,*,処理,ショリ,ショリ
の      助詞,連体化,*,*,*,*,の,ノ,ノ
うち    名詞,非自立,副詞可能,*,*,*,うち,ウチ,ウチ
で      助詞,格助詞,一般,*,*,*,で,デ,デ
(以下,继续输出)
```

使用 -O Wakati 选项，可以只执行单词的切分

```
C:\Users\odaka\ch2>extraction < text1.txt | mecab -O wakati
自然 言語 処理 の うち で 基本 的 かつ 実用 的 な 技術 は 、自然 言語 テキスト
の 入力 と 編集 を 支援 する 技術 でしょ う 。

C:\Users\odaka\ch2>type text2.txt
すももももももものうち
C:\Users\odaka\ch2>mecab < text2.txt
すもも  名詞,一般,*,*,*,*,すもも,スモモ,スモモ
も      助詞,係助詞,*,*,*,*,も,モ,モ
もも    名詞,一般,*,*,*,*,もも,モモ,モモ
も      助詞,係助詞,*,*,*,*,も,モ,モ
もも    名詞,一般,*,*,*,*,もも,モモ,モモ
の      助詞,連体化,*,*,*,*,の,ノ,ノ
うち    名詞,非自立,副詞可能,*,*,*,うち,ウチ,ウチ
EOS
C:\Users\odaka\ch2>
```

即使是相同的连续文字种类，也能够切分出单词（形态素）

2. 2-gram 分析

到目前为止，我们讲述了多种切分单词的方法，其结果可以用来分析单词 *n*-gram。其过程和文字 *n*-gram 的分析是相同的。下面给出其处理过程。

① 由单词 1-gram 生成单词 2-gram（tow2gram.c 程序）

↓

② 按词典顺序整理 2-gram（sort 命令）

↓

③ 将相同的 2-gram 汇总到一行中，将其重复次数添加到行首（uniqc.c 程序）

↓

④ 将行首的重复次数按降序排列（sortn.c 程序）

实例2.10给出了按照上述过程统计单词2-gram频率的示例。图中给出的是对本书开头部分文本的分析结果。其中频率最高的是日语文本句子末尾的“です－＞。”及图记号的一部分等，在其后出现的是“自然言語処理技術－＞の”、“の－＞抽出”、“の－＞技術”及“対話応答－＞システム”等2-gram特征。

实例2.10 单词2-gram频率分析示例[⊖]。

```
C:\Users\odaka\ch2>makew1gram <ch11.txt | tow2gram | sort | uniqc | sortn
7       1 -> .
7       です -> 。
6       図 -> 1
5       文書 -> の
4       インタフェ -> ー
4       ワ -> ー
4       ー -> ジ
4       。 -> また
4       自然言語処理技術 -> の
4       ー -> ス
4       ペ -> ー
4       ー -> ドプロセッサ
4         -> 自然言語処理
4       、 -> 文書
3       による -> 情報検索
3       音声 -> による
3       の -> 技術
3       の -> 入力
3       は -> 、
3       と -> 編集
3       。 -> 検索
3       自然言語 -> インタフェ
3       文書同士 -> の
3       また -> 、
(下面继续输出)
```

在上述处理过程中，步骤①采用tow2gram.c程序由单词1-gram生成单词2-gram。tow2gram.c的源代码和执行示例在清单2.5和实例2.11中给出。

⊖ 实际执行时，同一频率的单词2-gram出现的顺序和原书略有不同，这里给出的是译者实际执行的结果。——译者注

清单 2.5 tow2gram.c 程序的源代码。

```
/*************************************************/
/*            tow2gram.c                         */
/*  由makew1gram.c程序的输出                     */
/*  结果生成单词2-gram                           */
/* 使用方法                                      */
/*C:\Users\odaka\ch2>makew1gram<t.txt|tow2gram    */
/*************************************************/

/*和Visual Studio的互换性保证 */
#define _CRT_SECURE_NO_WARNINGS

/*include头文件*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/* 符号常数的定义 */
#define N 256 /*单词1-gram的字节长*/

/****************/
/* main()函数   */
/****************/
int main()
{
 char w1[N]="",w2[N]="" ;     /*输入的单词(1-gram)*/

/*读入数据后输出单词2-gram*/
 while(scanf("%s",w2)!=EOF){
  printf("%s->%s\n",w1,w2) ;
  strncpy(w1,w2,N) ;          /* 保存输入的单词 */
 }
return 0 ;
```

实例 2.11 tow2gram.c 程序的执行示例。

```
C:\Users\odaka\ch2>makew1gram <ch11.txt | tow2gram
->
->自然言語処理
自然言語処理->(
()は->、
、->コンピュ
コンピュ->ー
```

```
一 -> タプログラム
タプログラム -> を
を -> 用
用 -> いて
いて -> 自然言語
自然言語 -> を
を -> 処理
処理 -> する
する -> 技術
技術 -> です
です ->。
(下面继续输出)
```

2.1.3 1-of-*N*表示的处理

进行到现在的准备工作已经实现了从自然语言文本中切分出单词的处理过程。接下来，为了能将单词作为神经网络的输入，考虑将单词变换为1-of-*N*表示的方法，然后介绍从1-of-*N*表示变换为bag-of-words表示的方法。

1. 1-of-*N*表示

上一节所完成的makew1gram.c程序能够从文本中提取单词。为了用神经网络来处理提取出的单词，必须将单词这一非数值型数据用数值来表示。下面考虑将单词变换为1-of-*N*表示的方法。如第1章所述，1-of-*N*表示是通过以数值为元素的向量表示的，可以直接作为后面讲述的神经网络的输入使用。

为了实现1-of-*N*表示，首先需要对作为分析对象的文本中所含有的单词进行分类统计，得到单词种类的清单和总的单词种类数。然后再一次读入单词，求得每一个单词所对应的1-of-*N*表示向量。

考虑图2.7中给出的例子。其中①是作为分析对象的文本，针对这一文本使用makew1gram.c程序切分出单词。然后去除重复的单词，再将输入文本中所含有的单词集合起来，得到图中所示的45个不同种类的单词。顺便说一下，如果算上标点符号，原始文章中所含的单词数是76个。

由于单词的种类是 45 个，1-of-*N* 表示所采用的向量元素个数就是 45。然后，将每个单词在向量中所对应的位置设为 1，其他位置设为 0，其结果如图 2-7 中的②所示。

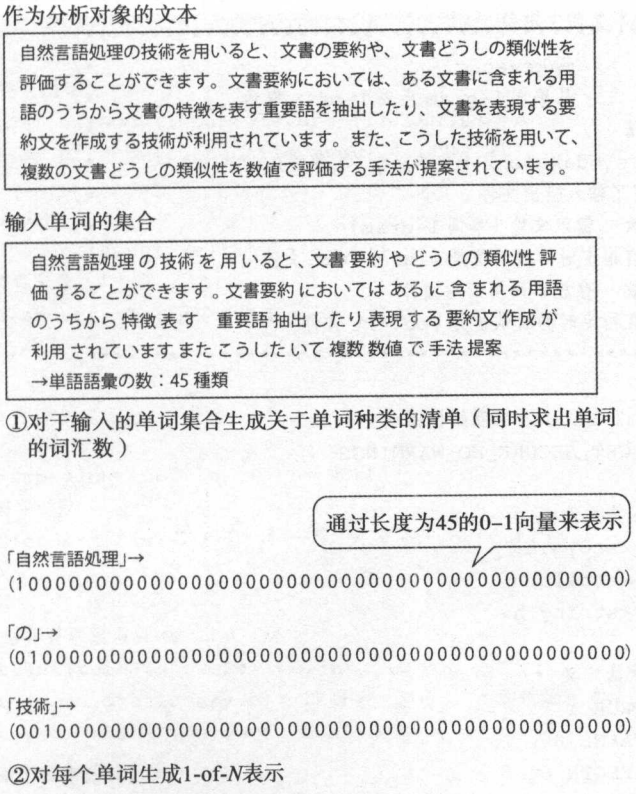


图 2.7　从单词的集合生成 1-of-*N* 表示

下面给出的是上述处理过程的总结。

生成单词种类的清单（录入单词词典中）

　读入单词 1-gram

　　如果是新出现的单词，则录入单词词典中

生成 1-of-*N* 表示

　读入单词 1-gram

　　仅在对应的位置设为 1（其他位置设为 0），输出该向量

清单 2.6 给出了按上述过程编写的 makevec.c 程序。

清单 2.6 makevec.c 程序。

```
/*************************************************/
/*          makevec.c                            */
/*          从单词 1-gram 生成 1-of-N 表示       */
/* 使用方法                                       */
/*C:\Users\odaka\ch2>makevec  (参数)               */
/* 参数指定了输入输出文件                        */
/* 第一参数  输入文件(单词 1-gram)               */
/* 参数没有指定时的默认值是 w1gram.txt 文件      */
/* 第二参数  存放单词词汇的文件                  */
/* 参数没有指定时的默认值是 voc.txt 文件         */
/*************************************************/

/* 和 Visual Studio 的互换性保证 */
#define _CRT_SECURE_NO_WARNINGS

/*include 头文件 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/* 符号常数的定义 */
#define TRUE 1
#define FALSE 0
#define LENGTH 64                          /* 单词长度的上限 */
#define N 5000                             /* 单词种类的上限 */
#define INPUTFILE "w1gram.txt"             /* 默认输入文件 */
#define OUTPUTFILE "voc.txt"               /* 默认输出文件 */

/* 函数原型的声明 */
int isnew(char word[],
      char dictionary[][LENGTH],int n);    /* 新建单词判别 */
void putvec(char word[],
      char dictionary[][LENGTH],int n);    /*1-of-N 输出 */

/****************/
/* main() 函数   */
/****************/
int main(int argc,char *argv[])
{
 char word[LENGTH*10] ;/* 用于读入单词 */
```

```
41  char dictionary[N][LENGTH] ;                    /* 用于录入单词的词典 */
42  int n=0 ;                                       /* 单词的种类数 */
43  FILE *fpi,*fpo ;                                /* 文件指针 */
44  char inputfile[LENGTH]=INPUTFILE ;              /* 输入文件名 */
45  char outputfile[LENGTH]=OUTPUTFILE ;            /* 输出文件名 */
46
47  /* 输入文件名的设定 */
48  if(argc>=2) strncpy(inputfile,argv[1],LENGTH) ;
49  if(argc>=3) strncpy(outputfile,argv[2],LENGTH) ;
50
51  /* 打开输入文件 */
52  if((fpi=fopen(inputfile,"r"))==NULL){
53   /* 打开文件失败 */
54   fprintf(stderr,"%s: ファイルオープン失敗 \n",inputfile) ;
55   exit(1);
56  }
57
58  /* 打开输出文件 */
59  if((fpo=fopen(outputfile,"w"))==NULL){
60   /* 打开文件失败 */
61   fprintf(stderr,"%s: ファイルオープン失敗 \n",outputfile) ;
62   exit(1);
63  }
64  /* 读入数据并登录到词典 */
65  while(fscanf(fpi,"%s",word)!=EOF){
66   if(isnew(word,dictionary,n)==TRUE){          /* 如果是新单词 */
67     strncpy(dictionary[n],word,LENGTH);        /* 录入单词 */
68     ++n ;/* 计数单词个数 */
69   }
70  }
71  fprintf(stderr," 単語数 %d\n",n) ;
72  rewind(fpi) ;                                   /* 向文件头回卷 */
73
74  /* 输出 1-of-N 表示 */
75  while(fscanf(fpi,"%s",word)!=EOF){
76   putvec(word,dictionary,n) ;                    /* 输出 */
77  }
78
79  /* 词汇文件的输出 */
80  {
81   int i ;
82   for(i=0;i<n;++i) fprintf(fpo,"%s\n",dictionary[i]) ;
83  }
84  return 0 ;
```

```
85 }
86
87 /*******************************/
88 /*  putvec() 函数                */
89 /* 输出 1-of-N 表示              */
90 /*******************************/
91 void putvec(char word[],
92       char dictionary[][LENGTH],int n)
93 {
94  int i ;              /* 循环控制变量 */
95
96  for(i=0;i<n;++i){
97   if((strncmp(word,dictionary[i],LENGTH)==0)
98        && (strlen(word)==strlen(dictionary[i]))) /* 一致 */
99    printf("1") ;
100   else printf("0") ;
101   printf(" ") ;
102  }
103  printf("\n") ;       /* 向量的区分 */
104 }
105
106 /*******************************/
107 /*  isnew() 函数                 */
108 /* 是否是新单词的判定            */
109 /*******************************/
110 int isnew(char word[],
111       char dictionary[][LENGTH],int n)
112 {
113  int i ;              /* 循环控制变量 */
114
115  for(i=0;i<n;++i){
116   if((strncmp(word,dictionary[i],LENGTH)==0)
117       && (strlen(word)==strlen(dictionary[i]))) break; /* 已经录入 */
118  }
119  if(i<n) return FALSE ;
120
121  return TRUE ;
122 }
```

到目前为止，所编写的程序中，基本上采用了标准的输入输出来实现数据的传送。因此，在 makevec.c 程序中也加入了标准的输入输出，其中一部分数据的传送是借助文件的介入而实现的。表 2.2 给出了 makevec.c 程序中使用的文件。

表 2.2 makevec.c 程序中使用的文件

文　件	默认文件名	说　明
单词 1-gram 输入文件	w1gram.txt	是 1-of-*N* 表示的来源。将单词通过换行或空白等分开后写入的文件。如果要指定默认文件名以外的文件，通过第一个命令行参数来指定
存放单词词汇的输出文件	voc.txt	删除重复的单词，按照单词出现的顺序排列并输出的文件。如果要指定默认文件名（voc.txt）以外的文件，通过第二个命令行参数来指定

实例 2.12 为 makevec.c 程序的运行示例，其中对于由两句话组成的简单文本生成其 1-of-*N* 表示。这里用 makew1gram.c 程序来提取存放原始文本的 text4.txt 文件中的单词，将其结果保存在 w1gram.txt 文件中，随后用 makevec.c 程序来生成文本的 1-of-*N* 表示。

makevec.c 程序输出的是单词的 1-of-*N* 表示，该表示是按其在文本中的出现顺序排列的，并给出了删除重复单词后按出现顺序排列的词汇清单。程序按照标准输出来输出 1-of-*N* 表示，因此，为了方便以后的应用，需要对其进行适当的重定向并保存到文件中。在实例 2.12 中，将程序的输出重定向到名为 makevecout.txt 的文件中保存起来。词汇清单可保存到指定的文件中，在实例 2.12 中，它被保存到了默认文件 voc.txt 中。

实例 2.12 makevec.c 程序的运行示例（1）。

```
C:\Users\odaka\ch2>type text4.txt
私は人工の知能、人工知能です。私は知能を有します。
C:\Users\odaka\ch2>makew1gram < text4.txt

私
は
人工
の
知能
、
人工知能
です
。
私
は
```

由自然语言记录的处理对象文本（输入文本）

使用 makew1gram.c 程序提取单词

```
知能
を
有
します
。
C:\Users\odaka\ch2>makew1gram < text4.txt > w1gram.txt

C:\Users\odaka\ch2>makevec
単語数 12
1 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0 0 0 0 0
0 0 0 0 0 0 1 0 0 0 0 0
0 0 0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0 1 0 0 0
1 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0 1 0 0 0
C:\Users\odaka\ch2>makevec > makevecout.txt
単語数 12

C:\Users\odaka\ch2>type voc.txt
私
は
人工
の
知能
、
人工知能
です
。
を
有
します
C:\Users\odaka\ch2>
```

将makew1gram.c程序的输出结果存放到w1gram.txt文件中

使用makevec.c程序生成1-of-*N*表示

将makevec.c程序的输出（1-of-*N*表示）存放到makevecout.txt中

由makevec.c程序生成的词汇一览（存放到voc.txt文件）

相对简单文本的运行结果

实例2.12对于非常短的文本生成了1-of-*N*表示。在实例2.13中，将本书第1章的

文本部分提取出来生成1-of-*N*表示。输入文本是由约40KB的日语文本组成的数据，由大约9000个单词组成。执行和实例2.12同样的处理后，其结果为1876元素的向量表示。由于这一输出结果非常庞大，实例2.13仅给出了其中一部分结果。

实例2.13 makevec.c程序的运行示例（2）。

```
c:\Users\odaka\ch2>type ch1.txt
第1章 自然言語処理と深層学習(40)

この章では、はじめに自然言語処理とは何かについて述べ、自然言語処理を実現するため
にどのような研究が進められてきたかを概観します。次に機械学習の一手法である深層学
習について、どのような技術であるのかをいくつかの例を通して説明します。これらを踏
まえた上で、自然言語処理と深層学習の関係を説明します。
(下面继续输出)

c:\Users\odaka\ch2>makew1gram < ch1.txt > ch1w1gram.txt

c:\Users\odaka\ch2>makevec ch1w1gram.txt ch1voc.txt > ch1vec.txt
単語数 1872

c:\User\odaka\ch2>type ch1vec.txt | C:\Windows\System32\more
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
(下面继续输出)
```

本书第1章的文本部分

由1876个元素构成的1-of-*N*表示（仅给出一部分）

具有一定规模文本的运行结果

1-of-*N*表示是为了作为神经网络的输入数据而使用，对于人类来说，这种表示形式并不是那么容易理解。因此，有必要将1-of-*N*表示变换为人类容易理解的自然语言文本表示。为此，本书给出了从1-of-*N*表示向自然语言文本进行逆变换的makes.c程序。

如实例 2.14 所示，makes.c 是输入 1-of-*N* 表示并输出相应自然语言文本的程序。在实例 2.14 中，实例 2.12 生成的 makevecout.txt 文件中所含的 1-of-*N* 表示被再变换为原始的日语文本。

实例 2.14 makes.c 程序的执行示例。

```
C:\Users\odaka\ch2>type makevecout.txt
1 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0 0 0 0 0
(下面继续输出 1-of-N 表示)

C:\Users\odaka\ch2>makes < makevecout.txt
単語数 12
私は人工の知能、人工知能です。私は知能を有します。
C:\User\odaka\ch2>
```

在 makevecout.txt 文件中存放 1-of-*N* 表示的数据

根据 makevecout.txt 文件和存放单词词汇的 voc.txt 文件生成原始的自然语言文本

makes.c 程序对 1-of-*N* 表示进行再变换时，需要知道向量的各元素是如何表示单词的。这里，makes.c 程序使用了 makevec.c 程序输出的单词词汇文件。makes.c 程序的处理过程如图 2.8 所示，将 1-of-*N* 表示的向量数据和词汇数据匹配起来，按顺序输出对应的单词。

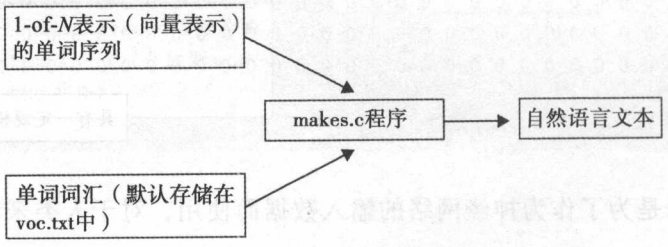


图 2.8 makes.c 程序的处理过程

makes.c 程序使用了表 2.3 中的文件进行处理。

表 2.3 makes.c 程序所使用的文件

文 件	默认文件名	说 明
单词词汇	voc.txt	存放单词词汇的文件。如果要指定默认文件名以外的文件，通过第一个命令行参数来指定
1-of-*N* 表示（向量表示）	（标准输入）	存放作为输入的 1-of-*N* 表示的数据，通过第二个命令行参数来指定。省略指定文件名时，通过标准输入读入数据

清单 2.7 给出了 makes.c 程序。

清单 2.7 makes.c 程序。

```
/*****************************************/
/*            makes.c                    */
/* 由 1-of-N 表示生成文本                 */
/* 使用方法                               */
/* C:\Users\odaka\ch2>makes(参数)          */
/* 参数指定了输入文件                     */
/* 第一参数  存放单词词汇的文件           */
/* 参数没有指定时的默认值是 voc.txt 文件  */
/* 第二参数  输入文件(1-of-N 表示)        */
/* 参数没有指定时按标准输入               */
/*****************************************/

/* 和 Visual Studio 的互换性保证 */
#define _CRT_SECURE_NO_WARNINGS

/*include 头文件 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/* 符号常数的定义 */
#define TRUE 1
#define FALSE 0
#define LENGTH 64              /* 单词长度的上限 */
#define N 10000                /* 单词种类的上限 */
#define VOCFILE "voc.txt"      /* 默认单词词汇文件 */

/***************/
/*  main() 函数  */
/***************/
int main(int argc,char *argv[])
```

```
{
 char word[LENGTH*10] ;                    /* 用于读入的单词 */
 char dictionary[N][LENGTH] ;              /* 用于登录单词的词典 */
 int n=0 ;                                 /* 单词的种类数 */
 FILE *fpvoc,*fpi ;                        /* 文件指针 */
 char inputfile[LENGTH] ;                  /* 输入文件名 */
 char vocfile[LENGTH]=VOCFILE ;            /* 单词词汇文件名 */
 int e ;                                   /* 输入向量元素值 */
 int i ;                                   /* 循环控制 */

 /* 输入文件名的设定 */
 fpi=stdin ;
 if(argc>=2) strncpy(vocfile,argv[1],LENGTH) ;
 if(argc>=3) strncpy(inputfile,argv[2],LENGTH) ;

 /* 打开词汇文件 */
 if((fpvoc=fopen(vocfile,"r"))==NULL){
  /* 打开文件失败 */
  fprintf(stderr,"%s: ファイルオープン失敗 \n",vocfile) ;
  exit(1);
 }

 /* 打开输入文件 */
 if((argc>=3)&&(fpi=fopen(inputfile,"r"))==NULL){
  /* 打开文件失败 */
  fprintf(stderr,"%s: ファイルオープン失敗 \n",inputfile) ;
  exit(1);
 }

 /* 读入单词词汇数据并登录到词典 */
 while(fscanf(fpvoc,"%s",word)!=EOF){
  strncpy(dictionary[n],word,LENGTH) ;      /* 登录单词 */
  ++n ;/* 计数单词个数 */
 }
 printf(" 単語数 %d\n",n) ;

 /* 输出文本 */
 i=0 ;
 while(fscanf(fpi,"%d",&e)!=EOF){
  if(e==1) printf("%s",dictionary[i]) ;
  ++i ;
  if(i>=n) i=0 ;
 }

 return 0 ;
}
```

上面给出了对 1-of-*N* 表示和自然语言文本进行相互转换的程序，图 2.9 总结了这些程序所使用的数据文件间的关系。

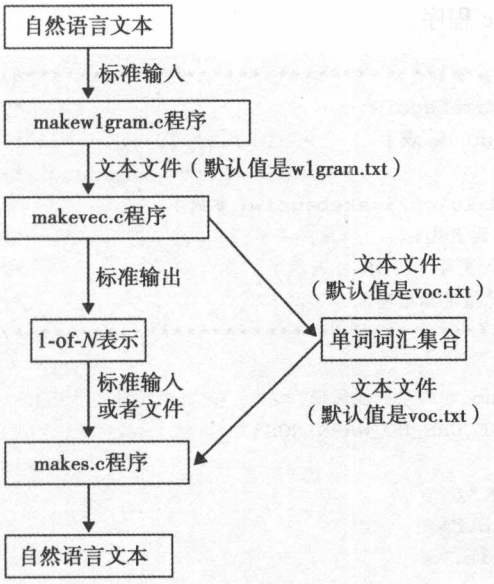


图 2.9　1-of-*N* 表示和自然语言文本之间的相互变换程序与各个子程序的关系

2. bag-of-words 表示

得到 1-of-*N* 表示后，就可以生成 bag-of-words 表示——将多个以 1-of-*N* 表示的数据的各元素相加合并，生成 bag-of-words 表示（图 2.10）。

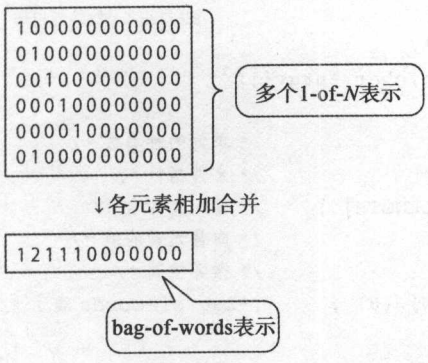


图 2.10　由 1-of-*N* 表示生成 bag-of-words 表示

程序 makebagofw.c 可用于读入多个 1-of-N 表示数据并生成 bag-of-words 表示数据，清单 2.8 是其源代码。其执行示例在实例 2.15 中给出。

清单 2.8 makebagofw.c 程序。

```
/*************************************************/
/*            makebagofw.c                       */
/* 生成 bag-of-words 表示                        */
/* 使用方法                                      */
/* C:\Users\odaka\ch2>makebagofw( 参数 )          */
/* 第一参数 单词词汇数 n                         */
/* 第二参数  输入文件 (1-of-N 表示 )             */
/* 参数没有指定时指定标准输入                    */
/*************************************************/

/* 和 Visual Studio 的互换性保证 */
#define _CRT_SECURE_NO_WARNINGS

/*include 头文件 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/* 符号常数的定义 */
#define TRUE 1
#define FALSE 0
#define N 1000                  /* 词汇 ( 单词 ) 种类的上限 */
#define LENGTH 64               /* 文件名长度的上限 */

/****************/
/*  main() 函数  */
/****************/
int main(int argc,char *argv[])
{
 int n ;                        /* 单词的种类数 */
 FILE *fpi ;                    /* 文件指针 */
 char inputfile[LENGTH] ;       /* 输入文件名 */
 int e ;                        /* 向量元素的值 */
 int i=0 ;                      /* 循环控制 */
 int bagofwords[N]={0} ;        /*bag-of-words 表示 */

 /* 由参数而设定初值 */
 if(argc<2){
```

```
 39   /* 参数个数不足 */
 40   fprintf(stderr," 使い方 \n >makenewvec "
 41           " 単語種類数 n  ( ファイル名 )\n") ;
 42   exit(1);
 43  }
 44  fpi=stdin ;                                    /* 默认是标准输入 */
 45  n=atoi(argv[1]) ;                              /* 设置词汇数 */
 46  if(argc>2){                                    /* 打开输入文件 */
 47   strncpy(inputfile,argv[2],LENGTH) ;           /* 输入文件 */
 48   if((fpi=fopen(inputfile,"r"))==NULL){
 49    /* 打开文件失败 */
 50    fprintf(stderr,"%s: ファイルオープン失敗 \n",inputfile) ;
 51    exit(1);
 52   }
 53  }
 54
 55  /* 读入基于 1-of-N 表示的单词序列 */
 56  while(fscanf(fpi,"%d",&e)!=EOF){
 57   bagofwords[i]+=e ;
 58   ++i ;
 59   if(i>=n) i=0 ;
 60  }
 61
 62  /* 输出 bag-of-words 表示 */
 63  for(i=0;i<n;++i)
 64   printf("%d ", bagofwords[i]);
 65  printf("\n") ;
 66
 67 return 0 ;
 68 }
```

实例 2.15 makebagofw.c 程序的执行示例。

```
C:\Users\odaka\ch2>type makevecout.txt
1 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0 0 0 0 0
0 0 0 0 0 0 1 0 0 0 0 0
0 0 0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0 1 0 0 0
1 0 0 0 0 0 0 0 0 0 0 0
```

```
0 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0 1 0 0 0
C:\Users\odaka\ch2>makebagofw 12 makevecout.txt
2 2 1 1 2 1 1 1 2 1 1 1
                          bag-of-words 表示（各元素相加合并的结果）
C:\Users\odaka\ch2>
```

2.2 基于单词 2-gram 的文本生成

本节将介绍利用 1-of-*N* 表示的单词序列来随机生成新单词序列的方法。该方法使用单词 2-gram 数据，按顺序将单词结合起来，生成新单词的词链，即生成新的文本。

这一方法采用的是单词 2-gram 数据。例如，图 2.11 给出了 12 种单词 2-gram。从“自然”这一单词开始，考虑使用单词 2-gram 适当串联单词的方法。例如，如②那样从“自然”开始，串联 1→5→6→7→8→9→10，就可以生成“自然言語処理技術が存在します（自然语言处理是存在的）”这一文本。同样，也是以“自然”开头，按照④的方式串联起来，就会生成“自然な技術があります（自然的技术是有的）”这一文本。

① 单词2-gram
1 自然->言語
2 自然->な
3 な->技術
4 人工->言語
5 言語->処理
6 処理->技術
7 技術->が
8 が->存在
9 存在->します
10 します->。
11 技術->があります
12 あります->。

图 2.11 通过 2-gram 生成单词序列

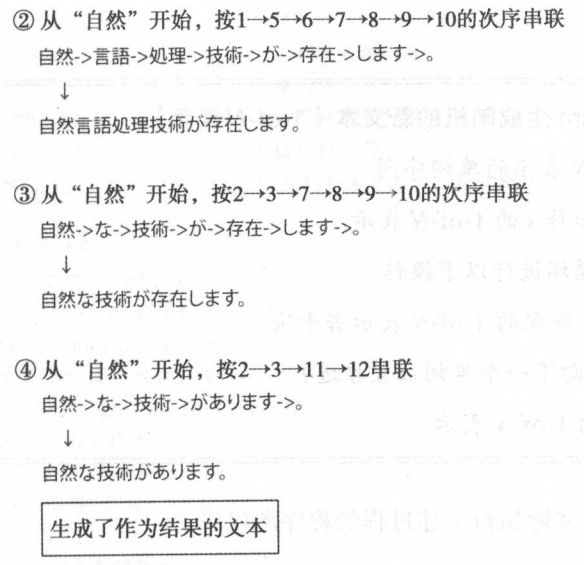


图 2.11 （续）

要用以上方法来生成文本，必须有单词 2-gram。这里可将前面完成的 makevec.c 程序输出的关于单词的 1-of-*N* 表示看作排成一列的单词 2-gram。

例如，在图 2.12 中，1-of-*N* 表示的第一行和第二行表示了文本开头的单词“我 -> 是”2-gram，第二行和第三行表示了“是 -> 人工”2-gram。这样一来，作为 makevec.c 程序输出结果的 1-of-*N* 表示可作为单词 2-gram 的集合来处理。

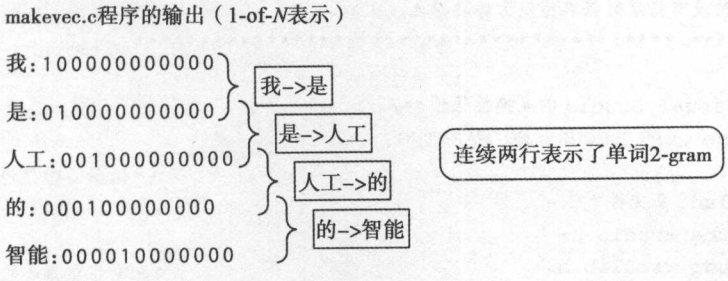


图 2.12 makevec.c 程序的输出（1-of-*N* 表示）可看作单词 2-gram

按照上述方法，读入 makevec.c 程序的输出，从作为起始符的单词开始，遵循单词 2-gram 按顺序进行单词的二项串联，就可以生成单词的词链。具体来说，所执行的处理

如下：

使用单词 2-gram 生成随机的新文本（1-of-*N* 表示）

读入基于 1-of-*N* 表示的单词序列

输出对应于起始符 s 的 1-of-*N* 表示

按适当的次数循环进行以下操作

　随机搜寻和 s 匹配的 1-of-*N* 表示若干次

　将 s 增加 1（向下一个单词位置推进）

　输出 s 对应的 1-of-*N* 表示

清单 2.9 给出了实际执行上述过程的程序源代码。

清单 2.9 makenewvec.c 程序。

```
/*************************************************/
/*              makenewvec.c                     */
/* 随机生成新文本(1-of-N表示)                     */
/* 使用方法                                       */
/* C:\Users\odaka\ch2>makenewvec(参数)             */
/*      参数指定输入文件                          */
/* 第一参数  单词词汇数 n                         */
/* 第二参数  起始符 s (0<=s<n)                    */
/* 参数没有指定时将其指定为0                      */
/* 第三参数 输入文件(1-of-N表示)                  */
/* 参数没有指定时将其指定为标准输入               */
/*************************************************/

/*和Visual Studio的互换性保证 */
#define _CRT_SECURE_NO_WARNINGS

/*include头文件*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/* 符号常数的定义 */
#define TRUE 1
#define FALSE 0
```

```
25 #define N 1000                  /* 词汇（单词）种类的上限 */
26 #define WN 5000                 /* 单词个数的上限 */
27 #define WLIMIT 50               /* 输出单词数 */
28 #define LENGTH 64               /* 文字列的长度的上限 */
29 #define SEED 65535              /* 随机数的种子 */
30 #define ULIMIT 5                /* 随机搜索单词的上限次数 */
31
32 /* 函数原型的声明 */
33 int read1ofn(FILE *fpi,int n) ;
34    /* 读入基于1-of-N表示的单词序列 */
35 void putvec(int nextn,int n) ;
36    /* 输出基于1-of-N表示的单词 */
37 int searchs(int s,int n,int wn) ;
38    /* 搜寻对应于s的单词 */
39 int matchptn(int i,int s,int n) ;
40   /* 检查单词的一致性                */
41 int rndn(int n) ;
42   /* 返回参数以下的整数随机数 */
43
44 /* 外部变量 */
45 char ngram[WN][N] ;             /* 存放1-of-N表示的输入数据 */
46
47 /*****************/
48 /*  main()函数   */
49 /*****************/
50 int main(int argc,char *argv[])
51 {
52  int n ;                        /* 单词的种类数 */
53  int wn ;                       /* 输入单词的总数 */
54  int s=0 ;                      /* 开始单词序号 */
55  FILE *fpi ;                    /* 文件指针 */
56  char inputfile[LENGTH] ;       /* 输入文件名 */
57  int i,j ;                      /* 循环控制 */
58  int loopmax ;                  /* 循环次数 */
59
60  /* 随机数的初始化 */
61  srand(SEED) ;
62
63  /* 根据参数设定初值 */
64  if(argc<2){
65   /* 参数不足 */
66   fprintf(stderr,"使い方 \n >makenewvec "
67           "単語種類数n（開始単語番号s）（ファイル名）\n") ;
68   exit(1);
```

```
69  }
70  fpi=stdin ;                                    /* 默认为标准输入 */
71  n=atoi(argv[1]) ;                              /* 设定词汇数 */
72  if(argc>2) s=atoi(argv[2]) ;                   /* 开始词汇序号 */
73  if(argc>3){                                    /* 打开输入文件 */
74   strncpy(inputfile,argv[3],LENGTH) ;           /* 输入文件 */
75   if((fpi=fopen(inputfile,"r"))==NULL){
76    /* 文件打开失败 */
77    fprintf(stderr,"%s: ファイルオープン失敗 \n",inputfile) ;
78    exit(1);
79   }
80  }
81
82  fprintf(stderr," 単語数 %d, 開始単語番号 %d\n",n,s) ;
83  if((s>=n)||(s<0)){                             /*S 不适当 */
84    fprintf(stderr,"s=%d,s は 1 以上 n 未満にしてください \n",s) ;
85    exit(1);
86  }
87  /* 读入基于 1-of-N 表示的单词序列 */
88  wn= read1ofn(fpi,n) ;                          /* 设定读入和输入单词总数 wn*/
89
90  /* 生成单词词链 ( 文本 )*/
91  putvec(s,n) ;                                  /* 起始符 */
92
93  for(i=0;i<WLIMIT;++i){
94   /* 搜寻对应于 s 的单词 */
95   loopmax=rndn(ULIMIT) ;                        /* 最大重复次数 ULIMIT*/
96   for(j=0;j<loopmax;++j)                        /* 重复随机次数 */
97    s=searchs(s,n,wn) ;
98   /* 输出相邻的下一个单词 */
99   ++ s ;
100  if(s>=wn) s=0 ;                                /* 回到最开始 */
101  putvec(s,n) ;
102 }
103
104 return 0 ;
105 }
106
107 /*************************************/
108 /*    rndn() 函数                    */
109 /* 返回参数以下的整数随机数          */
110 /*************************************/
111 int rndn(int n)
112 {
```

```
113  double rndno ;                          /* 生成的随机数 */
114
115  while((rndno=(double)rand()/RAND_MAX)==1.0) ;
116  return rndno*n ;
117 }
118
119 /*************************************/
120 /*   searcha() 函数                    */
121 /*  搜索 s 对应的单词                  */
122 /*************************************/
123 int searchs(int s,int n,int wn)
124 {
125  int i ;                                /* 循环控制 */
126
127 for(i=s+1;i<wn;++i)
128   if(matchptn(i,s,n)==TRUE) return i ;
129  for(i=0;i<=s;++i)
130   if(matchptn(i,s,n)==TRUE) return i ;
131  /* 都不一致 */
132  fprintf(stderr,"内部エラー  searchs() 関数 \n") ;
133  exit(1) ;
134 }
135
136 /*************************************/
137 /*   matchptn() 函数                   */
138 /* 检查单词的一致性                    */
139 /*************************************/
140 int matchptn(int i,int s,int n)
141 {
142  int result=TRUE ;
143  int index ;
144
145  for(index=0;index<n;++index)
146   if(ngram[i][index]!=ngram[s][index])
147    result=FALSE ;                       /* 不一致 */
148  return result ;
149 }
150
151 /*************************************/
152 /*  putvec() 函数                      */
153 /* 输出基于 1-of-N 表示的单词          */
154 /*************************************/
155 void putvec(int nextn,int n)
156 {
```

```
157  int j=0 ;                    /* 循环控制 */
158
159  for(j=0;j<n;++j)
160   printf("%1d ",ngram[nextn][j]);
161  printf("\n") ;
162
163 }
164
165 /*****************************************/
166 /*   read1ofn() 函数                     */
167 /* 读入基于 1-of-N 表示的单词序列        */
168 /*****************************************/
169 int read1ofn(FILE *fpi,int n)
170 {
171  int e ;              /* 输入向量元素的值 */
172  int i=0,j=0 ;        /* 循环控制 */
173
174  while((fscanf(fpi,"%d",&e)!=EOF)&&(i<WN)){
175   ngram[i][j]=e ;
176   ++j ;
177   if(j>=n){
178    j=0 ; ++i ;
179   }
180  }
181  return i ;
182 }
```

实例 2.16 给出了 makenewvec.c 程序的执行示例，是一个基于简单输入数据的例子。在该例中，使用了含有 12 个词汇（单词数）的例文来生成具有 50 个词语的词链。由于很难理解 1-of-N 表示的输出，本例也给出了使用 makes.c 程序变换回原始自然语言文本的结果。在实行例 2.16 的例子中，词汇和 2-gram 的数量都比较少，虽然希望输出和输入内容相同的文本，但输出文本中出现了在输入文本中没有的“私は人工の知能が有します(我有人工的智能)”这样的文本。

实例 2.16 makenewvec.c 程序的执行示例（1）。

```
C:\Users\odaka\ch2>type text4.txt
私は人工の知能、人工知能です。私は知能を有します。
C:\Users\odaka\ch2>makenewvec 12 0 makevecout.txt
単語数 12, 開始単語番号 0
1 0 0 0 0 0 0 0 0 0 0 0
```

原始文本数据（单词词汇数 12）

```
0 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0 1 0 0 0
1 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0 0 0
(下面继续输出)

C:\Users\odaka\ch2>makenewvec 12 0 makevecout.txt | makes
単語数 12, 開始単語番号 0
単語数 12
私は知能を有します。私は人工の知能、人工知能です。私は人工の知能を有します。私は
人工の知能、人工知能です。私は知能を有します。私は人工の知能、人工知能です。私
C:\Users\odaka\ch2>
```

从起始符 0（“私（我）”）开始文本生成的结果

实例 2.17 也是 makenewvec.c 程序的执行示例，这一次输入数据的词汇数是 45。和实例 2.16 的示例相比，在实例 2.17 中能看到更为丰富的文本。乍一看，这里得到的文本就像普通的文章，但由于在这一文本生成的结果中没有考虑对文本意义的处理，这一段文本从意义上讲存在很多问题。此外，如果将作为生成文本出发点的单词（起始符）变更一下，就会生成不同的文本，实例 2.17 也给出了这样的例子。

实例 2.17 makenewvec.c 程序的执行示例（2）。

原始文章数据（单词词汇数 45）

```
C:\Users\odaka\ch2>type text3.txt
  自然言語処理の技術を用いると、文書の要約や、文書どうしの類似性を評価することができます。
文書要約においては、ある文書に含まれ用語のうちから文書の特徴を表す重要語を抽出したり、文
書を表現する要約文を作成する技術が利用されています。また、こうした技術を用いて、複数の文
書どうしの類似性を数値で評価する手法が提案されています。

C:\Users\odaka\ch2>makenewvec 45 0 makevecout.txt | makes
単語数 45, 開始単語番号 0
単語数 45
  自然言語処理の特徴を表す重要語を数値で評価する手法が利用されています。また、こ
うした技術を表現する要約文を用いて、文書どうしの類似性を作成する手法が利用されて
います。  自然言語処理の技術を表現する手法が提案

C:\User\odaka\ch2>makenewvec 45 3 makevecout.txt | makes
```

起始符（单词）为 0（“自然言语处理”）时的输出

```
単語数 45, 開始単語番号 3
単語数 45
技術を評価することができます。文書要約においては、こうした技術を評価することができます。文書要約においては、こうした技術が利用されています。文書要約においては、文書どうしの類似性を評価することができます。文書要約においては、文書の特徴を用いると、文書どうしの類似性を用いると、こうした
```

起始符（单词）为 3（“技术”）时的输出

实例 2.18 是另一个文本生成的示例。和实例 2.17 相比，实例 2.18 增加了更多的单词词汇，生成文本时用了 299 个单词词汇。

实例 2.18 makenewvec.c 程序的执行示例（3）。

原始文本数据的一部分（单词词汇数 299）

```
c:\Users\odaka\ch2>type text5.txt
この章では、はじめに自然言語処理とは何かについて述べ、自然言語処理を実現するためにどのような研究が進められてきたかを概観します。次に機械学習の一手法で…
（以下，继续输出）

c:\Users\odaka\ch2>makenewvec 299 0 makevecout.txt | makes
単語数 299, 開始単語番号 0
単語数 299
この章では、途中まで入力した検索語を検索や、綴りのミスを検索したり表記の応用事例です。日本語入力を編集することもできます。例えば語の典型的な利用方法の一つでしょう。　対話システムのもあります。これらのシステム用いて自然言語による

c:\Users\odaka\ch2>makenewvec 299 20 makevecout.txt | makes
単語数 299, 開始単語番号 20
単語数 299
概観します。　機械に用いられるだけでなく、ある文書の道具として用いられています。これらは、入力にしたがって情報を提供する場合だけでなく、調べたい単語や整理？保存などの特定の応用から生まれたものです。例を通して表示することが可能です
```

起始符（单词）序号为 0（“这个”）时的输出

起始符（单词）序号为 20（“概观”）时的输出

CHAPTER 3

第 3 章

深度学习应用于自然语言文本分析

本章讲述将深度学习应用于自然语言文本分析的方法示例，采用了卷积神经网络这一深度学习方法。具体而言，是用第 2 章的方法生成单词序列的 1-of-*N* 表示，然后将其输入到卷积神经网络中，用卷积神经网络对该单词序列进行评估。

3.1 基于 CNN 的文本分类

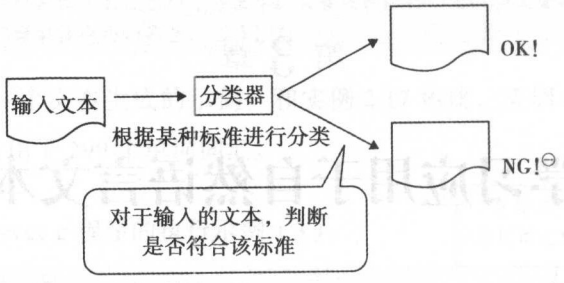
在自然语言处理应用中，常常需要基于某种尺度对文本进行分类。例如，在用自然语言处理技术进行文本校正时，需要遵从某种标准对文本进行分类，判断所给的文本是否是正确的。或者，考虑从多个文本中自动提取其所属的类别，在某种基准下对各式文本中含有的文本进行分类，就可自动提取同一类别下的文书（图 3.1）。

这里考虑用**卷积神经网络**（以下称为 CNN）对文本进行分类、评估，也就是说，在图 3.1 中采用 CNN 来构造分类器。

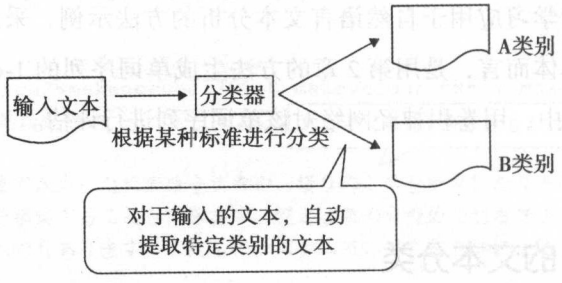
如第 1 章所述，CNN 是具有强大图像识别能力的神经网络。这里将单词序列的 1-of-*N* 表示输入到 CNN 中，可以实现对输入单词序列的识别。

图 3.2 为基于 CNN 进行分类学习的框架。首先，如图①所示，为了能够输入到 CNN 中，需要将由自然语言记录的文本变换为 1-of-*N* 表示。其次，作为 CNN 学习的准

备，要创建学习数据集。学习数据集由两部分组成，一部分是由 1-of-*N* 表示的文本，另一部分是遵从某种规范的、希望获得的文本分类结果——教师（标签）数据。



①文本校正



② 提取相同类别的文本

图 3.1 文本自动分类的应用

作为教师数据，既可以用单一的数值来表示输入的单词序列是否属于某个类别，也可以用多个数值来表示在多类别情形下如何分类。这里为了简单起见，将教师数据设为单一的数值，其取值在 0 和 1 之间，表示单词序列在多大程度上属于某个类别。

输入到 CNN 的数据由 1-of-*N* 表示的单词序列及对之评估的教师数据组合而成，多个这样的数据组合构成了学习数据。这样就完成了 CNN 学习的准备工作。

图 3.2 中的③是 CNN 的学习过程。在这一阶段，将学习数据集中 1-of-*N* 表示的文本输入到 CNN 中，调整网络参数，使得 CNN 的输出向教师数据（评估值）逼近。

⊖ Not Good 的缩写。——译者注

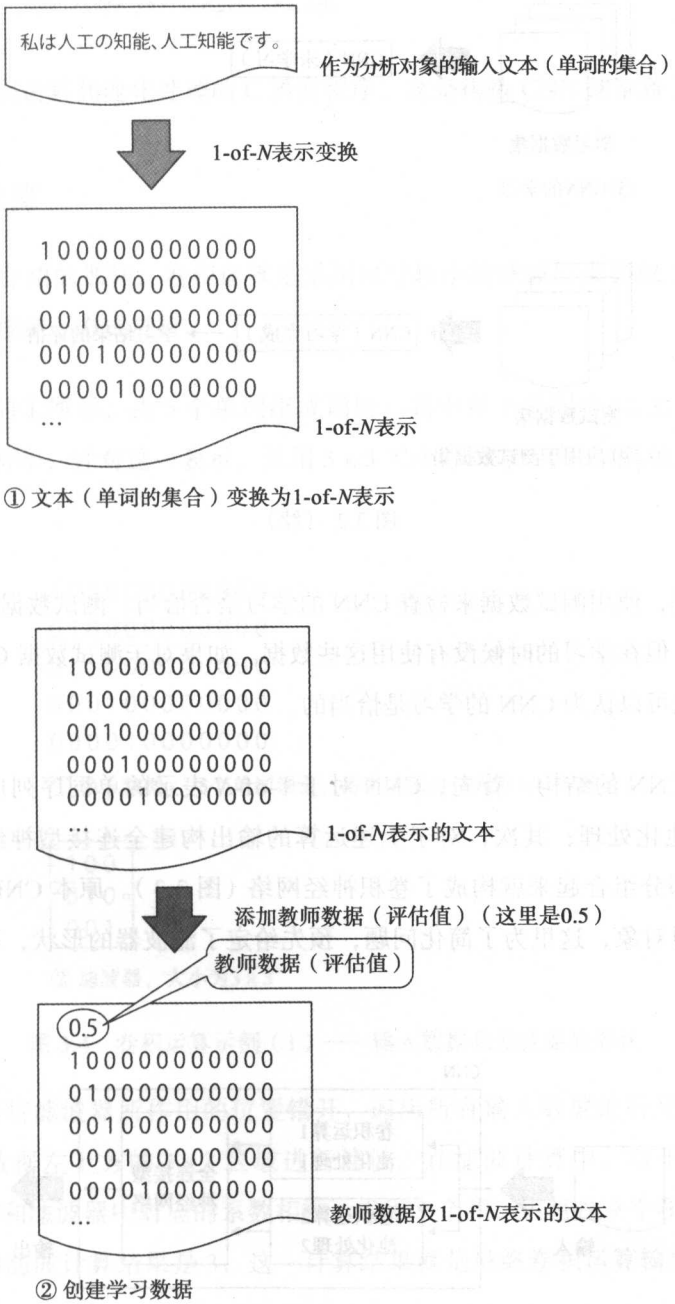


图 3.2　基于 CNN 进行分类学习的框架

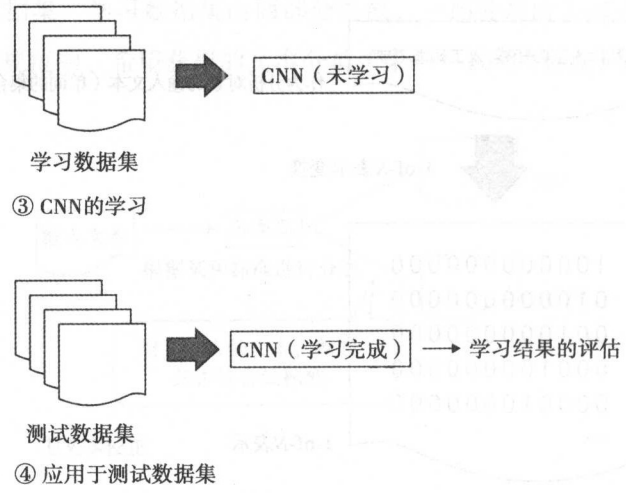


图 3.2 （续）

学习结束后，使用测试数据来检查 CNN 的学习是否恰当。测试数据的格式和学习数据的格式相同，但在学习的时候没有使用这些数据。如果对于测试数据 CNN 能够输出所期待的结果，就可以认为 CNN 的学习是恰当的。

下面介绍 CNN 的结构。首先，CNN 对于 1-of-*N* 表示的单词序列应用滤波方法进行卷积运算和池化处理；其次，对于上述运算的输出构建全连接型神经网络来做进一步处理。这两部分组合起来就构成了卷积神经网络（图 3.3）。原本 CNN 中卷积滤波器的参数也是学习对象，这里为了简化问题，预先给定了滤波器的形状，不将其作为学习对象。

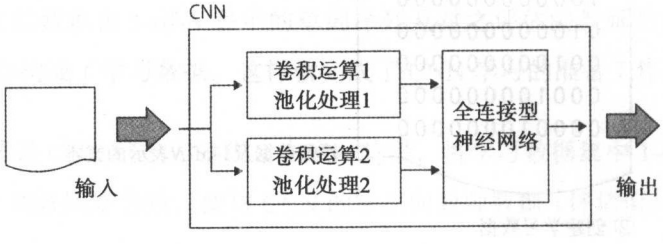


图 3.3 CNN 的结构

3.2 准备 1：卷积运算和池化处理

本节实现卷积运算和池化处理的 C 语言程序，这是构造 CNN 的准备工作。

3.2.1 卷积运算

正如前面所介绍的那样，卷积运算是采用尺寸较小的滤波器来提取二维输入数据的特征的。其计算按如下方式进行。

如图 3.4 中的①所示，由 5 个单词组成词链，其中每个单词由 12 元素 1-of-*N* 表示。如图 3.4 中的②所示，针对这一表示，采用 3×3 大小、斜方向为 1（其他元素为 0）的滤波器。

```
100000000000
010000000000
001000000000
000100000000
000010000000
```

① 输入数据， 5个单词词链，每个单词由12元素1-of-*N*表示

```
100
010
001
```

② 滤波器，大小为3×3

图 3.4 卷积运算示例（1）——输入数据和滤波器的形状

卷积运算是将滤波器所作用的位置错开，遍历所有输入数据进行反复计算。例如，首先对于输入数据左上方的 3×3 区域进行滤波。在滤波计算中，对于 3×3 这 9 个数据，将输入数据和滤波器中对应的系数相乘，求得 9 个积，再将这 9 个积相加。如图 3.5 所示，左上方的滤波计算结果是 3，这一计算结果就是最终卷积运算输出结果中左上方的值。

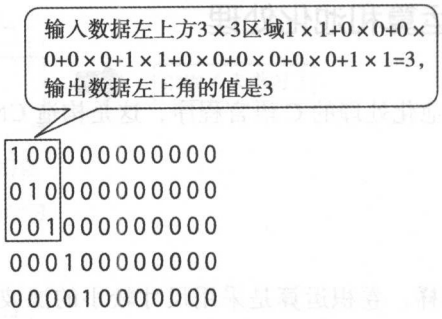


图 3.5 卷积运算示例（2）——在输入数据左上方应用滤波器，求得其结果

下面将滤波器的作用位置向右移动 1 位。这样一来，如图 3.6 所示，其卷积运算的输出是 0。其结果是输出数据中从左边开始第 2 个元素的值是 0。

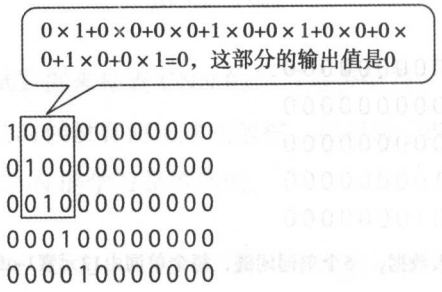


图 3.6 卷积运算示例（3）——在下一个位置应用滤波器，求得其结果

将以上的滤波计算横向重复 10 次，纵向重复 3 次，就对 1-of-*N* 表示的全体输入数据进行了滤波处理，将所有滤波结果集中起来，就得到了卷积运算的输出结果。

下面看一个卷积运算的例子。图 3.7 中，在由 1-of-*N* 表示的单词数据上，值为 1 的元素沿斜方向排列。这说明该文本中首次出现的单词是按顺序排列起来的（顺接关系）。对于这样的文本，应用图中带有斜方向元素的滤波器，就能够强化并提取这一特征。变更滤波器的形状，使其具有纵向的元素，应用这样的滤波器不但不能强化文本特征，反而使得其特征变得模糊。

下面在 1-of-*N* 表示的单词数据上，考虑数值为 1 的元素纵向排列的情形（图 3.8）。这一情形表示的是在文本中相同单词反复出现（连续关系）。对于这样的情况，应用纵向

有元素的滤波器，能够强化并提取这样的特征。如果采用斜方向有元素的滤波器，则不能强化特征，反而使特征变得模糊。

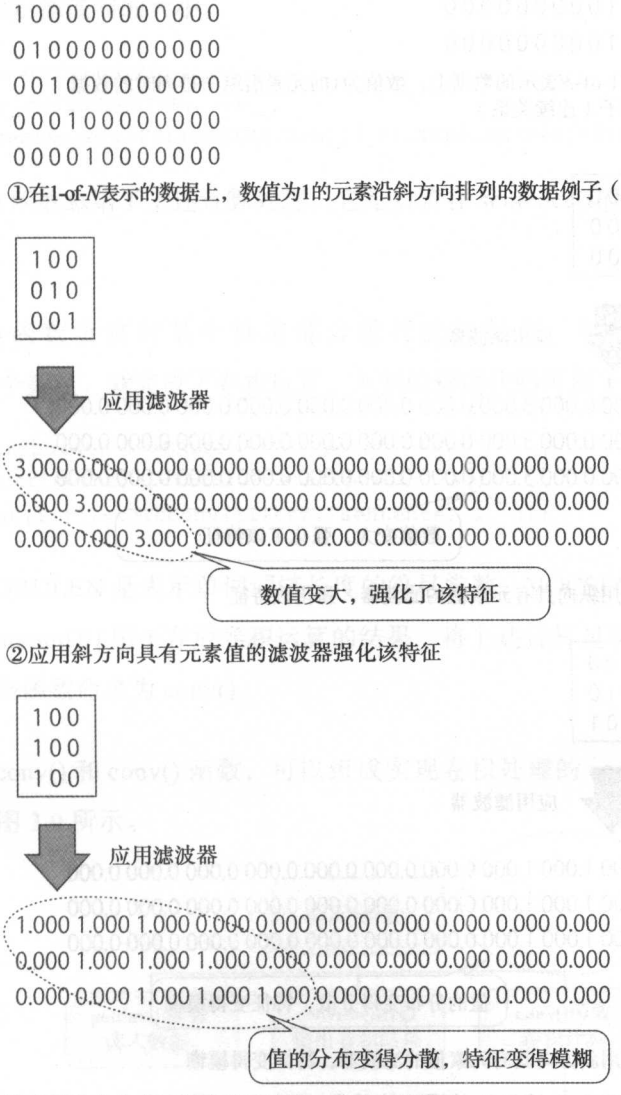


图 3.7　由卷积运算提取单词连接状态（1）——顺接关系

像上面这样，对于 1-of-*N* 表示的单词序列施以卷积运算，能够提取出原始单词序列的特征。

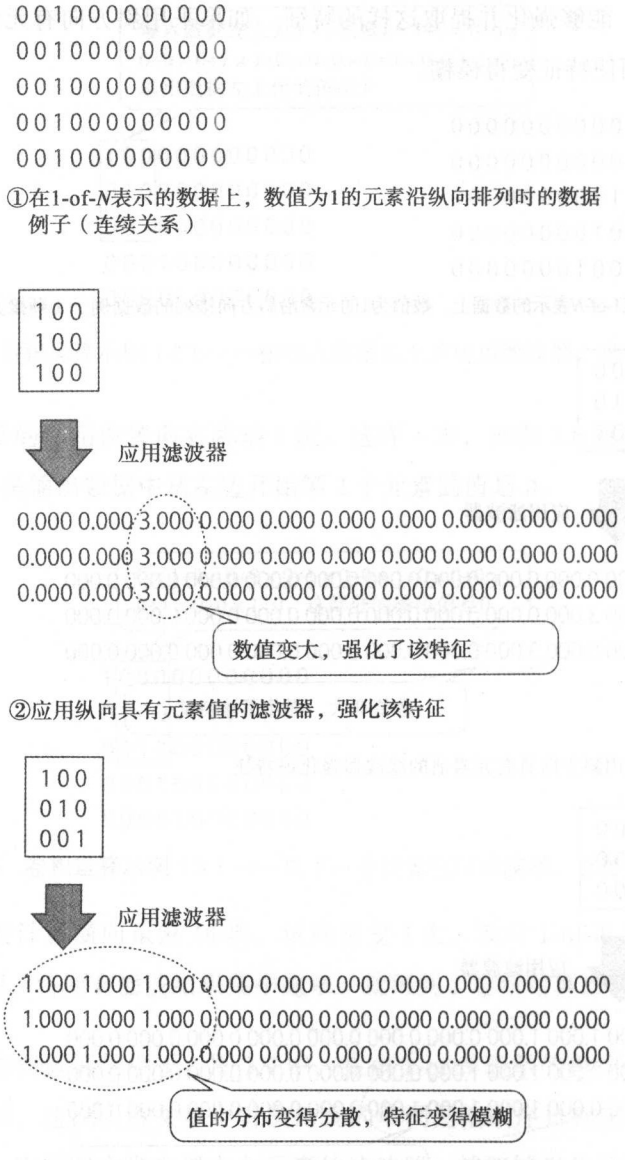


图 3.8　由卷积运算提取单词连接状态（2）——同一单词的连续关系

那么，如何才能用程序来实现卷积运算呢？卷积运算要反复进行乘法运算和加法运算，因此其计算只是单纯的积、和的计算。因而，只需在数组中存入输入数据和滤波器系数，不断按顺序计算其值即可。

将输入数据放置在二维数组 sentence[][] 中。如下所示，对于某元素 sentence[i][j]，将其与对应的数组 filter[][] 中存放的系数相乘，再进行相加合并计算。这里的符号常数 FILTERSIZE 表示滤波器的大小。

```
for(m=0; m<FILTERSIZE; ++m)
  for(n=0; n<FILTERSIZE; ++n)
    sum+=sentence[i-FILTERSIZE/2+m][j-FILTERSIZE/2+n]*filter[m][n];
```

calcconv() 函数总结了上述计算处理，它是负责在 sentence[i][j] 周围进行滤波处理的函数。

calcconv() 函数负责对某个特定部分进行滤波处理，因此，如果将其作用于 sentence[][] 整个数组，就完成了卷积运算。下面的程序代码可用于实现这一目的。

```
for(i=startpoint; i<WORDLEN-startpoint; ++i)
  for(j=startpoint; j<VOCSIZE-startpoint; ++j)
    convout[i][j]=calcconv(filter, sentence, I, j);
```

其中，WORDLEN 是表示单词词链长度的符号常数，VOCSIZE 表示词汇的种类数。此外，数组 convout[][] 用于存放卷积运算的结果。将上述计算过程总结为负责卷积运算的函数，并将该函数命名为 conv()。

使用 calcconv() 和 conv() 函数，可以组成实现卷积处理的 conv.c 程序。conv.c 程序的内部结构如图 3.9 所示。

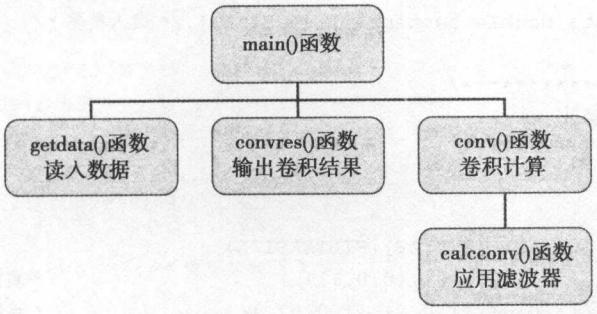


图 3.9 conv.c 程序的内部结构

有了上述准备工作，下面来编写 conv.c 程序。清单 3.1 给出了 conv.c 程序的源代码。

清单 3.1 conv.c 程序。

```
/*************************************************************/
/*                        conv.c                             */
/*  卷积处理                                                 */
/*  读入 1-of-N 数据，实施卷积                               */
/* 使用方法                                                  */
/*  \Users\odaka\ch3>conv  < data1.txt                       */
/*************************************************************/

/* 和 Visual Studio 的互换性保证 */
#define _CRT_SECURE_NO_WARNINGS

/*include 头文件 */
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

/* 符号常数的定义 */
#define VOCSIZE 12          /*1-of-N 表示的词汇数（次数）*/
#define WORDLEN 5           /*1-of-N 表示的单词词链长度 */
#define FILTERSIZE 3        /* 滤波器的大小 */

/* 函数原型的声明 */
void conv(double filter[][FILTERSIZE]
   ,double sentence[][VOCSIZE]
   ,double convout[][VOCSIZE]); /* 卷积计算 */
double calcconv(double filter[][FILTERSIZE]
               ,double sentence[][VOCSIZE], int i, int j);
/* 应用滤波器 */
void convres(double convout[][VOCSIZE]); /* 输出卷积结果 */
void getdata(double sentence[][VOCSIZE]);/* 读入数据 */

/*******************/
/* main() 函数 */
/*******************/
int main()
{
 double filter[FILTERSIZE][FILTERSIZE]
      ={{1,0,0},{0,1,0},{0,0,1}} ;                    /* 顺接滤波 */
//      = {{1,0,0},{1,0,0},{1,0,0} };                  /* 连续滤波 */
 double sentence[WORDLEN][VOCSIZE];                   /* 输入数据 */
 double convout[WORDLEN][VOCSIZE] = { 0 };            /* 卷积输出 */

```

```
43
44  /* 读入输入数据 */
45  getdata(sentence);
46
47  /* 卷积计算 */
48  conv(filter, sentence, convout);
49
50  /* 输出结果 */
51  convres(convout);
52
53  return 0;
54 }
55
56 /**********************/
57 /* convres() 函数 */
58 /* 输出卷积结果 */
59 /**********************/
60 void convres(double convout[][VOCSIZE])
61 {
62  int i, j;                              /* 循环控制 */
63  int startpoint = FILTERSIZE/2;         /* 输出范围的下限 */
64
65  for (i = startpoint; i<WORDLEN - 1; ++i){
66   for (j = startpoint; j<VOCSIZE - 1; ++j){
67    printf("%.3lf ", convout[i][j]);
68   }
69   printf("\n");
70  }
71  printf("\n");
72 }
73
74 /**********************/
75 /*   getdata() 函数     */
76 /* 读入输入数据          */
77 /**********************/
78 void getdata(double e[][VOCSIZE])
79 {
80  int i = 0, j = 0;     /* 循环控制用 */
81
82  /* 输入数据 */
83  while (scanf("%lf", &e[i][j]) != EOF){
84   ++j;
```

```
 85   if (j >= VOCSIZE){          /* 下一个数据 */
 86    j = 0;
 87    ++i;
 88    if (i >= WORDLEN) break; /* 输入结束 */
 89   }
 90  }
 91 }
 92
 93 /**********************/
 94 /*  conv() 函数        */
 95 /*  卷积计算           */
 96 /**********************/
 97 void conv(double filter[][FILTERSIZE]
 98          ,double sentence[][VOCSIZE], double convout[][VOCSIZE])
 99 {
100  int i = 0, j = 0;                  /* 循环控制用 */
101  int startpoint = FILTERSIZE / 2;   /* 卷积范围的下限 */
102
103  for (i = startpoint; i<WORDLEN - startpoint; ++i)
104   for (j = startpoint; j<VOCSIZE - startpoint; ++j)
105    convout[i][j] = calcconv(filter, sentence, i, j);
106 }
107
108 /**********************/
109 /*  calcconv() 函数    */
110 /*   应用滤波器        */
111 /**********************/
112 double calcconv(double filter[][FILTERSIZE]
113               , double sentence[][VOCSIZE], int i, int j)
114 {
115  int m, n;            /* 循环控制用 */
116  double sum = 0;      /* 和的值 */
117
118  for (m = 0; m<FILTERSIZE; ++m)
119   for (n = 0; n<FILTERSIZE; ++n)
120    sum += sentence[i - FILTERSIZE/2 + m][j - FILTERSIZE / 2 + n]*filter[m][n];
121
122  return sum;
123 }
```

实例 3.1 给出了 conv.c 程序的执行示例。其中，从存放 1-of-*N* 表示的单词序列文件 makevecout.txt 中读取数据，并采用两种不同类的滤波器来进行卷积运算。

实例 3.1 conv.c 程序的执行示例。

```
C:\User\odaka\ch3>type makevecout.txt
1 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0 0 0 0 0
(以下省略)

C:\User\odaka\ch3>conv < makevecout.txt
3.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 3.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 3.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000

C:\User\odaka\ch3>
(1) 顺接滤波器的应用示例

C:\User\odaka\ch3>conv < makevecout.txt
1.000 1.000 1.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 1.000 1.000 1.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 1.000 1.000 1.000 0.000 0.000 0.000 0.000 0.000

C:\User\odaka\ch3>
(2) 连续滤波器的应用示例
```

单词顺接排列

在 conv.c 程序中使用顺接滤波器

特征被强化了

使用连续滤波器

特征变得模糊

在实例 3.1 中，（1）和（2）都是通过 conv.c 程序来执行的，但实际上这两个 conv.c 程序是不同的。也就是说，其内部所采用的滤波器形状是不同的。具体来说，在执行前，要将如下所示的 main() 函数开头部分的源代码的下画线部分进行修改后再重新编译。

在实例 3.1 中应用顺接滤波器时，可直接编译清单 3.1 的源代码。如果在实例 3.1 中采用连续滤波器，则需要对下面的下画线部分（通过注释来处理）进行修改。

```
32: /********************/
33: /*     main() 函数    */
34: /********************/
35: int main()
36: {
37: double filter[FILTERSIZE][FILTERSIZE]
```

```
38:         ={{1,0,0},{0,1,0},{0,0,1}} ;                    /* 顺接滤波 */
39://         = {{1,0,0}, {1,0,0}, {1,0,0}};                /* 连续滤波 */
```

将顺接滤波器注释掉，使用
连续滤波器来代替

```
38://         ={{1,0,0},{0,1,0},{0,0,1}} ;                    /* 顺接滤波 */
39:         = {{1,0,0}, {1,0,0}, {1,0,0}};                  /* 连续滤波 */
```

3.2.2 池化处理

为了实现CNN中卷积处理的基本构成要素，下面在conv.c程序上增加池化处理。池化处理可以采用和卷积处理相似的方法来实现。也就是说，将卷积运算中的滤波器换成从滤波器的作用范围内提取最大值或者计算平均值的滤波器。

可以根据下述处理来实现输出范围内最大值的“最大值池化”。

```
max=convout[i+POOLSIZE/2][j+POOLSIZE/2];
 for(m=i-POOLSIZE/2; m<=i+POOLSIZE/2;++m)
  for(n=j-POOLSIZE/2; n<=j+POOLSIZE/2;++n)
    if(max<convout[m][n]) max=convout[m][n];
```

表3.1给出了最大值池化处理中相关变量和符号常数的意义。表3.1中，convout[][]数组中存放的是要提取最大值的对象数据。最大值滤波器适用于以convout[i][j]为中心的边长为POOLSIZE的范围。求得这一范围内的最大值后，将其存放到变量max中。convpool.c程序实现了池化处理，maxpooling()函数对上述处理进行了总结。

表3.1 最大值池化处理中相关变量和符号常数的意义

名　称	说　明
max	存放范围内最大值的变量
convout[][]	存放要提取最大值的对象数据的数组
i,j	指定滤波器适用范围的坐标值
m,n	在滤波器适用范围内逐次指定的变量
POOLSIZE	滤波器的大小

在对象数据全范围内实施池化处理时，对应于对象数据的全范围遍历变量i,j的值，使用maxpooling()函数来实施池化。下面给出这一处理的代码。

```
int i, j;                                    /* 循环控制 */
int startpoint=FILTERSIZE/2+POOLSIZE/2;      /* 池化计算范围的下限 */

for(i=startpoint; i<WORDLEN-startpoint;++i)
 for(j=startpoint; j<VOCSIZE-startpoint;++j)
  poolout[i][j]=maxpooling(convout,i,j);
```

在 convpool.c 程序中将上述处理总结为 pool() 函数，并将其添加到 conv.c 程序中。这样一来，就有了如下所示的 convpool.c 程序的处理过程。

convpool.c 程序的处理过程

① getdata() 函数读取输入数据

② conv() 函数进行卷积计算

③ convres() 函数输出卷积计算的结果

④ pool() 函数进行池化计算

⑤ poolers() 函数输出结果

清单 3.2 给出了 convpool.c 程序的具体实现示例。

清单 3.2　convpool.c 程序。

```
/*************************************************************/
/*                      convpool.c                           */
/* 卷积和池化处理                                             */
/* 读入 1-of-N 数据，实施卷积和池化                           */
/* 使用方法                                                   */
/* \Users\odaka\ch3>convpool  < data1.txt                     */
/*************************************************************/

/* 和 Visual Studio 的互换性保证 */
#define _CRT_SECURE_NO_WARNINGS

/* include 头文件 */
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

/* 符号常数的定义 */
#define VOCSIZE 12          /*1-of-N 表示的词汇数（次数）*/
```

```
#define WORDLEN 7                /*1-of-N表示的单词的词链长度*/
#define FILTERSIZE 3             /*滤波器的大小*/
#define POOLSIZE 3               /*池化大小*/

/*函数原型的声明*/
void conv(double filter[][FILTERSIZE]
    , double sentence[][VOCSIZE]
    , double convout[][VOCSIZE]);              /*卷积计算*/
double calcconv(double filter[][FILTERSIZE]
                , double sentence[][VOCSIZE], int i, int j);
                                               /* 应用滤波器 */
void convres(double convout[][VOCSIZE]);       /*输出卷积结果*/
void getdata(double sentence[][VOCSIZE]);      /*读入数据*/
void poolres(double poolout[][VOCSIZE]);       /*输出池化结果*/
void pool(double convout[][VOCSIZE]
          , double poolout[][VOCSIZE]);        /*池化计算*/
double maxpooling(double convout[][VOCSIZE]
                  , int i, int j);             /* 最大值池化 */

/*******************/
/*    main()函数   */
/*******************/
int main()
{
double filter[FILTERSIZE][FILTERSIZE]
     ={{1,0,0},{0,1,0},{0,0,1}};               /*顺接滤波器*/
//      ={{1,0,0},{1,0,0},{1,0,0}} ;           /*连续滤波器*/
double sentence[WORDLEN][VOCSIZE];             /*输入数据*/
double convout[WORDLEN][VOCSIZE] = { 0 };      /*卷积输出*/
double poolout[WORDLEN][VOCSIZE] = { 0 };      /*输出数据*/

/*读入输入数据*/
getdata(sentence);

/*卷积计算*/
conv(filter, sentence, convout);

/*输出卷积计算的结果*/
convres(convout);

/*池化计算*/
pool(convout, poolout);

```

```
/* 输出结果 */
poolres(poolout);

return 0;
}

/***********************/
/*  poolres() 函数     */
/*   输出结果          */
/***********************/
void poolres(double poolout[][VOCSIZE])
{
int i, j;/* 循环控制 */
int startpoint = FILTERSIZE / 2 + POOLSIZE / 2; /* 池化计算范围的下限 */

for (i = startpoint; i<WORDLEN - startpoint; ++i){
 for (j = startpoint; j<VOCSIZE - startpoint; ++j)
  printf("%.3lf ", poolout[i][j]);
 printf("\n");
 }
 printf("\n");
}

/***********************/
/*  pool() 函数        */
/*     池化计算        */
/***********************/
void pool(double convout[][VOCSIZE]
        , double poolout[][VOCSIZE])
{
int i, j;                                          /* 循环控制 */
int startpoint = FILTERSIZE / 2 + POOLSIZE / 2;  /* 池化计算范围的下限 */

for (i = startpoint; i<WORDLEN - startpoint; ++i)
 for (j = startpoint; j<VOCSIZE - startpoint; ++j)
  poolout[i][j] = maxpooling(convout, i, j);
}

/*************************/
/* maxpooling() 函数     */
/* 最大值池化            */
/*************************/
double maxpooling(double convout[][VOCSIZE]
```

```
105                 , int i, int j)
106 {
107 int m, n;                    /* 循环控制用 */
108 double max;          /* 最大值 */
109
110 max=convout[i+POOLSIZE/2][j+POOLSIZE/2];
111 for(m=i-POOLSIZE/2; m<=i+POOLSIZE/2; ++m)
112  for(n=j-POOLSIZE/2; n<=j+POOLSIZE/2; ++n)
113   if(max<convout[m][n]) max = convout[m][n];
114
115  return max;
116 }
117
118 /**********************/
119 /*  convres() 函数      */
120 /* 输出卷积结果          */
121 /**********************/
122 void convres(double convout[][VOCSIZE])
123 {
124  int i, j;                                    /* 循环控制 */
125  vint startpoint = FILTERSIZE / 2;            /* 输出范围的下限 */
126
127 for (i = startpoint; i<WORDLEN - 1; ++i){
128  for (j = startpoint; j<VOCSIZE - 1; ++j){
129   printf("%.3lf ", convout[i][j]);
130  }
131  printf("\n");
132 }
133 printf("\n");
134 }
135
136 /**********************/
137 /*  getdata() 函数      */
138 /*  读入输入数据         */
139 /**********************/
140 void getdata(double e[][VOCSIZE])
141 {
142 int i = 0, j = 0;/* 循环控制用 */
143
144 /* 输入数据 */
145 while (scanf("%lf", &e[i][j]) != EOF){
146  ++j;
147  if (j >= VOCSIZE){/* 下一个数据 */
```

```
148   j = 0;
149   ++i;
150   if (i>=WORDLEN) break;/* 输入结束 */
151   }
152  }
153 }
154
155 /***********************/
156 /*  conv() 函数        */
157 /*  卷积计算           */
158 /***********************/
159 void conv(double filter[][FILTERSIZE]
160         , double sentence[][VOCSIZE], double convout[][VOCSIZE])
161 {
162  int i = 0, j = 0;                    /* 循环控制用 */
163  int startpoint = FILTERSIZE / 2;     /* 卷积范围的下限 */
164
165  for (i = startpoint; i<WORDLEN - startpoint; ++i)
166   for (j = startpoint; j<VOCSIZE - startpoint; ++j)
167    convout[i][j] = calcconv(filter, sentence, i, j);
168 }
169
170 /***********************/
171 /*  calcconv.c() 函数  */
172 /*  应用滤波器         */
173 /***********************/
174 double calcconv(double filter[][FILTERSIZE]
175             , double sentence[][VOCSIZE], int i, int j)
176 {
177  int m, n;              /* 循环控制用 */
178  double sum = 0;        /* 和的值 */
179
180  for (m = 0; m<FILTERSIZE; ++m)
181   for (n = 0; n<FILTERSIZE; ++n)
182    sum+=sentence[i-FILTERSIZE/2+m][j-FILTERSIZE/2+n]*filter[m][n];
183
184  return sum;
185 }
```

实例 3.2 给出了 convpool.c 程序的执行示例。在实例 3.2 中，对于卷积运算的结果实施最大值池化操作，输出卷积运算和池化处理的结果。

实例 3.2 convpool.c 程序的执行示例。

```
C:\User\odaka\ch3>convpool < makevecout.txt
3.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 3.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 3.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 3.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 3.000 0.000 0.000 0.000 0.000 0.000

3.000 3.000 3.000 0.000 0.000 0.000 0.000 0.000
3.000 3.000 3.000 3.000 0.000 0.000 0.000 0.000
3.000 3.000 3.000 3.000 3.000 0.000 0.000 0.000

C:\User\odaka\ch3>
```

卷积运算的输出结果

池化处理（最大值池化）的结果

3.3 准备 2：全连接型神经网络

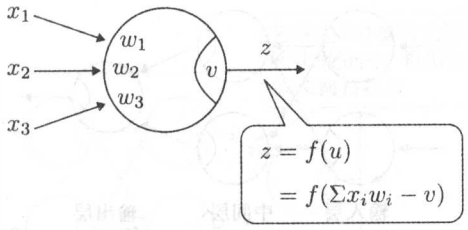
为了实现 CNN，这里介绍第二阶段的准备——构建全连接型神经网络。本节要介绍的基于层次结构的全连接型神经网络，是 CNN 和循环神经网络等各种神经网络的基本构成要素。

3.3.1 基于层次结构的全连接型神经网络的构造及学习方法

为了构建神经网络，必须实现作为其构成要素的**人工神经元**。

人工神经元是图 3.10 所示的计算元件。图中所示的是有 3 个输入和 1 个输出的人工神经元，并对于不同的输入定义了相对应的权重。人工神经元将输入数据和其对应的权重相乘后，将得到的积相加合并起来；然后，从合并值中减去一个常数（称为阈值）；接着，在减去阈值的结果上应用函数 f，函数 f 的输出就是人工神经元的输出。这里的函数 f 称为**传递函数（transfer function）**或**输出函数（output function）**。

其中，权重和阈值是人工神经元的参数，需要通过后面所讲述的学习方法来为其设定合适的值。



其中 $x_1 \sim x_3$ ：输入
$w_1 \sim w_3$ ：权重
v ：阈值
z ：输出

图 3.10 人工神经元的功能

sigmoid 函数（sigmoid function）是传递函数的一个例子，图 3.11 给出了该函数的形状。sigmoid 函数和反向传播算法的相容性非常好，因此，在使用反向传播算法时会经常采用 sigmoid 函数作为传递函数。

$$f(u) = \frac{1}{1 + e^{-u}}$$

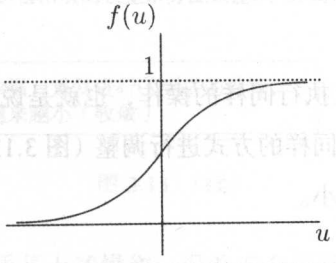


图 3.11 sigmoid 函数

将人工神经元组合起来可以构成神经网络。图 3.12 给出了一个简单的示例，图中所示的是一个 3 层神经网络，具有 2 个输入和 1 个输出，它是一个全连接型神经网络，层与层之间的神经元采用全连接方式结合在一起。其中，输入层的人工神经元将输入数据原封不动地向下一层传递，在中间层和输出层上进行实际的计算处理。无论在哪一层，人工神经元都进行加权求和以及传递函数的计算，并输出其结果。

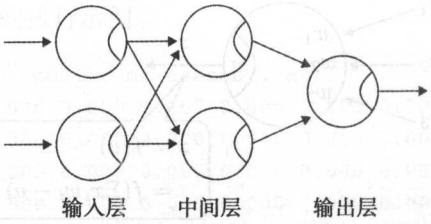


图 3.12 全连接型神经网络示例（2 输入、1 输出）

如第 1 章所述，图 3.12 所示的神经网络可以通过**反向传播**算法来学习。反向传播的原理如图 3.13 所示。

在图 3.13 的①中，假设计算得到的全体神经网络的输出结果为 o，作为教师数据的精确解为 o_t，o 和 o_t 不一致。此时，输出结果 o 和精确解 o_t 之间的差是输出误差 E。

考虑误差 E 产生的原因。最终输出 o 是输出层人工神经元的计算结果，而输出层人工神经元在计算时使用了输出层的输入值。此时，和输入值相对应的权重值越大，就会对产生误差 E 的原因产生更大的影响。因此，可以将误差 E 根据输出层权重按比例分配，由所分配的误差值来调整相对应的权重。经过这样的调整，误差 E 就会变得更小（图 3.13 中的②）。

在中间层的人工神经元上执行同样的操作，也就是说，根据输出层权重按比例分配的误差，用来对中间层权重按同样的方式进行调整（图 3.13 中的③）。这一来，误差 E 的绝对值肯定会比调整前要来得小。

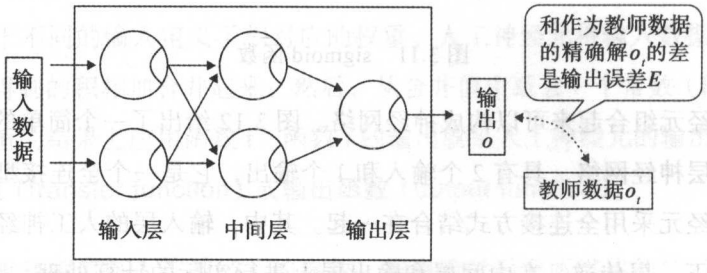
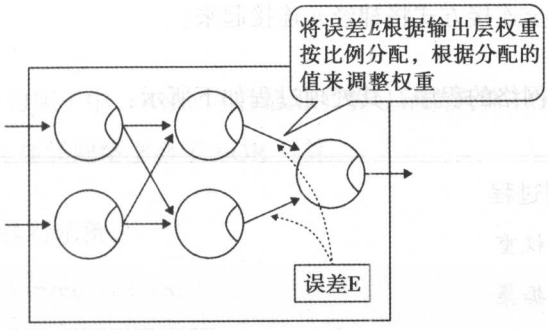
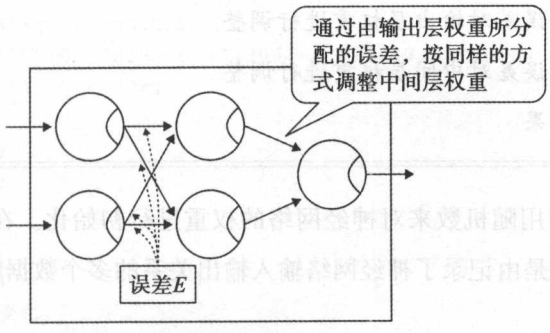


图 3.13 反向传播的原理



② 根据中间层到输出层的权重来分配误差E，并调整各自的权重



③ 根据中间层到输出层的权重来分配误差E，并调整各自的权重[⊖]

对各个学习数据，重复进行①到③的计算，
误差会变得越来越小（收敛）

图 3.13 （续）

此后，对于各个学习数据重复上述操作，误差 E 会逐渐变小。最终，当误差 E 下降到某个允许程度之下时，就可以终止学习的迭代过程了。以上是基于反向传播的学习原理。

3.3.2 全连接型神经网络的实现

现在，我们实际构造层次型神经网络并实现基于误差反向传播的学习程序。在进一步学习神经网络实现过程时，请记住图 3.12 中所示的 3 层、2 输入、1 输出的层次型神

⊖ 原文是根据中间层到输出层的权重来分配误差 E，并调整各自的权重。——译者注

经网络，在层间采用全连接方式将神经元连接起来。

bp.c 是实现神经网络的程序，其处理过程如下所示：

bp.c 程序的处理过程

（1）随机初始化权重

（2）读入学习数据集

（3）重复以下操作，直至误差降到一个给定值以下

①对于一个学习数据，计算其输出误差

②基于输出误差对输出层权重进行调整

③基于输出误差对中间层权重进行调整

（4）输出学习结果

在步骤 1 中，利用随机数来对神经网络的权重进行初始化。在步骤 2 中，读入学习数据集。学习数据集是由记录了神经网络输入输出关系的多个数据组成的集合。

步骤 3 是学习的主体。在步骤 3 中，从学习数据集中取出一个数据，利用该数据计算出神经网络的输出值，求出它和教师数据的差，这就是输出误差（步骤 3 中的①）。步骤 3 中的②和③则利用求出的输出误差，基于误差反向传播方法来调整权重。在学习过程中，对整个学习数据集执行上述步骤。

当误差降到一定程度之下时，终止学习，在步骤 4 中输出学习结果。学习结果包括所求得的权重以及对应于学习数据集的神经网络输出。

按上述方式所编写的 bp.c 程序较长，故将其放在附录 C 中。bp.c 程序的执行示例在实例 3.3 中给出。

在实例 3.3 中，首先将学习数据集 and.txt 文件输入到 bp.c 程序中。and.txt 文件中记录的是 AND 运算，即“逻辑与”所对应的输入、输出数据，这些数据将被作为神经网络的学习数据。将 and.txt 文件输入到 bp.c 程序中，经过 268 次迭代后神经网络学会了

AND 运算。

在实例 3.3 所给出的例子中，AND 之后给出了学习异或运算 EOR 的结果。在经过了 1202 次的迭代学习后，神经网络学会了 EOR 运算。

实例 3.3 bp.c 程序的执行示例。

```
C:\User\odaka\ch3>type and.txt
0 0 0
0 1 0
1 0 0
1 1 1
C:\User\odaka\ch3>bp < and.txt
 0.064486 0.440718 -0.108188 0.934996 -0.791437 0.399884
-0.875362 0.049715  0.991211
学习数据个数: 4
1       0.905330
2       0.921529
3       0.926556
4       0.926768
5       0.926443
(下面继续输出误差的变化)
265     0.001010
266     0.001006
267     0.001001
268     0.000996
-2.651725 -5.519571 -5.410115 4.835389 1.097955 3.205046
-8.056481 6.018151 1.423246
0 0.000000 0.000000 0.000000 0.000100
1 0.000000 1.000000 0.000000 0.010163
2 1.000000 0.000000 0.000000 0.018579
3 1.000000 1.000000 1.000000 0.976951

C:\User\odaka\ch3>type eor.txt
0 0 0
0 1 1
1 0 1
1 1 0
C:\User\odaka\ch3>bp < eor.txt
 0.064486 0.440718 -0.108188 0.934996 -0.791437 0.399884
-0.875362 0.049715  0.991211
学习数据个数: 4
```

AND 运算，即逻辑与对应的输入输出数据

迭代 268 次后学会了 AND 运算

EOR 运算，即异或运算对应的输入输出数据

```
1       1.975691
2       2.350580
3       2.172687
4       2.596810
5       2.040317
(下面继续输出误差的变化)
1210    0.001025
1211    0.001022
1212    0.001020
1213    0.001018
1214    0.001015
1215    0.001013
1216    0.001011
1217    0.001009
1218    0.001006
1219    0.001004
1220    0.001002
1221    0.000999
5.545034 -7.016814  2.693139 9.815056 -7.939982 -4.411229
9.282673 -9.016606 -4.346818
0 0.000000 0.000000 0.000000 0.018464
1 0.000000 1.000000 1.000000 0.983542
2 1.000000 0.000000 1.000000 0.983797
3 1.000000 1.000000 0.000000 0.010867

C:\Users\odaka\ch3>
```

迭代1221次后学会了EOR运算⊖

3.4 卷积神经网络的实现

3.4.1 卷积神经网络的结构

前面介绍了卷积运算、池化处理和全连接型神经网络，将这些部分组合起来就构成了卷积神经网络。因此，这里将执行卷积运算和池化处理操作的 convpool.c 和实现全连接型神经网络的 bp.c 程序组合起来，就能够实现卷积神经网络。

图 3.14 给出了卷积神经网络程序 cnn.c 的结构。可以看到，cnn.c 程序是由 convpool.c

⊖ 译者在 Microsoft Visual Studio Professional 2013 下编译执行该程序，和原书结果略有不同（原书迭代 1202 次后达到误差限）。——译者注

程序及 bp.c 程序组合而成的。

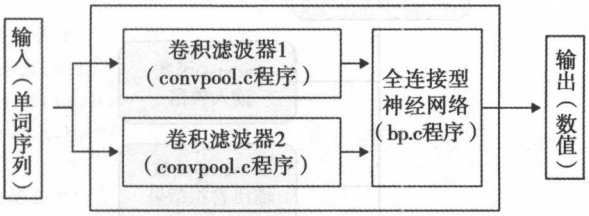


图 3.14 卷积神经网络程序 cnn.c 的结构

在图 3.14 的各个卷积滤波器中，对于按符号常数 VOCSIZE 和 WORDLEN 所规定范围内的由 1-of-*N* 表示的单词序列，执行卷积运算和池化处理。在这个程序中，给出了两种不同的卷积滤波器。卷积滤波器 1 用于检测单词的顺接关系，卷积滤波器 2 用于检测重复出现单词的连续关系。这些滤波器输出的数据个数如图 3.15 所示，在后面的全连接型神经网络中，这些数值将作为输入由该网络进行处理。

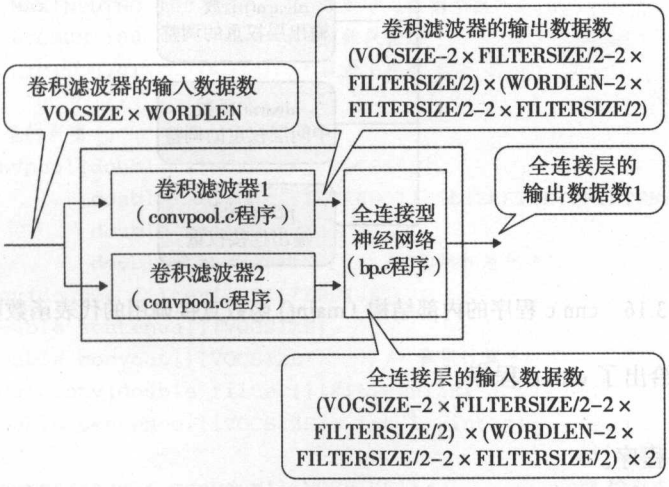


图 3.15 卷积滤波器和全连接型神经网络间的数据传输关系

3.4.2 由卷积神经网络学习 1-of-*N* 表示数据

cnn.c 程序是由 convpool.c 程序和 bp.c 程序组合而成的程序，其内部结构如图 3.16 所示。图 3.16 给出了 cnn.c 程序中由 main() 函数直接调用的代表性函数群。

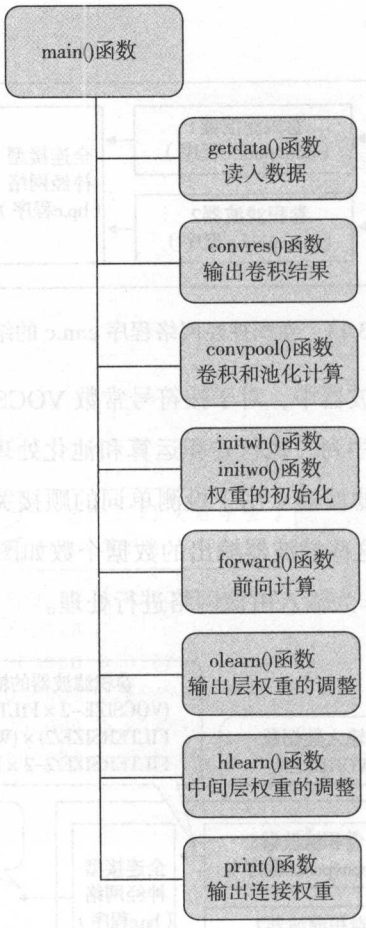


图 3.16 cnn.c 程序的内部结构（main() 函数直接调用的代表函数群）

清单 3.3 中给出了 cnn.c 程序。

清单 3.3 cnn.c 程序。

```
/*************************************************************/
/*                          cnn.c                            */
/* 进行卷积运算的神经网络                                      */
/* 使用方法                                                   */
/*  \Users\odaka\ch3>cnn < data.txt > result.txt              */
/*  输出误差的变化、学习结果中连接系数等                       */
/*************************************************************/

```

```
/* 和 Visual Studio 的互换性保证 */
#define _CRT_SECURE_NO_WARNINGS

/*include 头文件 */
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

/* 符号常数的定义 */
#define VOCSIZE 12              /*1-of-N 表示的词汇数 ( 次数 )*/
#define WORDLEN 7               /*1-of-N 表示的单词词链长度 */
#define FILTERSIZE 3            /* 滤波器的大小 */
#define POOLSIZE 3              /* 池化的大小 */
#define FILTERNO 2              /* 滤波器的个数 */

#define INPUTNO 48              /* 输入层神经元数 */
/* 由词汇数和单词词链长度确定 ((12-2-2)*(7-2-2))*FILTERNO*/
#define HIDDENNO 2              /* 中间层神经元数 */
#define ALPHA  10               /* 学习系数 */
#define SEED 7                  /* 随机数种子 */
#define MAXINPUTNO 100          /* 最大学习数据个数 */
#define BIGNUM 100              /* 初始误差值 */
#define LIMIT 0.01              /* 误差上限值 */

/* 函数原型的声明 */
void convpool(double s[WORDLEN][VOCSIZE],
              double mfilter[FILTERNO][FILTERSIZE][FILTERSIZE],
              double se[INPUTNO + 1],
              double teacher);      /* 卷积和池化 */
void conv(double filter[][FILTERSIZE]
    , double sentence[][VOCSIZE]
    , double convout[][VOCSIZE]);   /* 卷积计算 */
double calcconv(double filter[][FILTERSIZE]
    , double sentence[][VOCSIZE], int i, int j);
                                                    /* 应用滤波器 */
void convres(double convout[][VOCSIZE]);            /* 输出卷积结果 */
int getdata(double sentence[MAXINPUTNO][WORDLEN][VOCSIZE],
                  double teacher[MAXINPUTNO]);      /* 读入数据 */
void poolres(double poolout[][VOCSIZE]);            /* 输出池化结果 */
void pool(double convout[][VOCSIZE]
          , double poolout[][VOCSIZE]);             /* 池化计算 */
double maxpooling(double convout[][VOCSIZE]
                  , int i, int j);                  /* 最大值池化 */
double f(double u);                                 /* 传递函数 (sigmoid 函数)*/
```

```
53 void initwh(double wh[HIDDENNO][INPUTNO + 1]);
54                                                  /* 中间层权重的初始化 */
55 void initwo(double wo[HIDDENNO + 1]);           /* 输出层权重的初始化 */
56 double drnd(void);                              /* 生成随机数 */
57 void print(double wh[HIDDENNO][INPUTNO + 1]
58            , double wo[HIDDENNO + 1]);          /* 输出结果 */
59 double forward(double wh[HIDDENNO][INPUTNO + 1]
60          , double wo[HIDDENNO + 1], double hi[]
61          , double e[INPUTNO + 1]);  /* 前向计算 */
62 void olearn(double wo[HIDDENNO + 1], double hi[]
63          , double e[INPUTNO + 1], double o);   /* 输出层权重的调整 */
64 void hlearn(double wh[HIDDENNO][INPUTNO + 1]
65     , double wo[HIDDENNO + 1], double hi[]
66     , double e[INPUTNO + 1], double o);        /* 中间层权重的调整 */
67
68 /*******************/
69 /*   main() 函数    */
70 /*******************/
71 int main()
72 {
73  double mfilter[FILTERNO][FILTERSIZE][FILTERSIZE]
74        = {
75           { { 1, 0, 0 }, { 0, 1, 0 }, { 0, 0, 1 } },
76           { { 1, 0, 0 }, { 1, 0, 0 }, { 1, 0, 0 } }
77           };/* 滤波群 */
78 double sentence[MAXINPUTNO][WORDLEN][VOCSIZE];     /* 输入数据 */
79 double convout[WORDLEN][VOCSIZE] = { 0 };          /* 输出卷积结果 */
80 double poolout[WORDLEN][VOCSIZE] = { 0 };          /* 输出数据 */
81
82 double teacher[MAXINPUTNO];                       /* 教师数据 */
83 double wh[HIDDENNO][INPUTNO + 1];                 /* 中间层权重 */
84 double wo[HIDDENNO + 1];                          /* 输出层权重 */
85 double e[MAXINPUTNO][INPUTNO + 1];                /* 学习数据集 */
86 double hi[HIDDENNO + 1];                          /* 中间层权重 */
87 double o;                                         /* 输出 */
88 double err = BIGNUM;                              /* 误差评估 */
89 int i, j;                                         /* 循环控制用 */
90 int n_of_e;                                       /* 学习数据的个数 */
91 int count = 0;                                    /* 重复次数的计数 */
92
93 /* 初始化随机值 */
94 srand(SEED);
95
96 /* 初始化权重 */
```

```
 97 initwh(wh);              /* 中间层权重的初始化 */
 98 initwo(wo);              /* 输出层权重的初始化 */
 99 print(wh, wo);           /* 输出结果 */
100
101 /* 读入学习数据 */
102 n_of_e = getdata(sentence, teacher);
103 printf(" 学習データの個数 :%d\n", n_of_e);
104
105 /* 卷积和池化的计算 */
106 for (i = 0; i<n_of_e; ++i){
107  convpool(sentence[i], mfilter, e[i], teacher[i]);
108 }
109
110 /* 学习 */
111 while (err>LIMIT){
112  err = 0.0;
113  for (j = 0; j<n_of_e; ++j){
114   /* 前向计算 */
115   o = forward(wh, wo, hi, e[j]);
116   /* 输出值权重的调整 */
117   olearn(wo, hi, e[j], o);
118   /* 中间层权重的调整 */
119   hlearn(wh, wo, hi, e[j], o);
120   /* 计算误差的累积 */
121   err += (o - e[j][INPUTNO])*(o - e[j][INPUTNO]);
122  }
123  ++count;
124  /* 输出误差 */
125  fprintf(stderr, "%d\t%lf\n", count, err);
126 }/* 学习结束 */
127
128 /* 输出连接权重 */
129 print(wh, wo);
130
131 /* 输出对学习数据的计算结果 */
132 for (i = 0; i<n_of_e; ++i){
133  printf("%d\n", i);
134  for (j = 0; j<INPUTNO + 1; ++j).
135   printf("%lf ", e[i][j]);
136  printf("\n");
137  o = forward(wh, wo, hi, e[i]);
138  printf("%lf\n\n", o);
139 }
140
```

```
 return 0;
}

/**********************/
/*  poolres() 函数     */
/*   输出结果          */
/**********************/
void poolres(double poolout[][VOCSIZE])
{
 int i, j;                                          /* 循环控制用 */
 int startpoint = FILTERSIZE/2+POOLSIZE/2;          /* 池化计算范围的下限 */

 for (i = startpoint; i<WORDLEN - startpoint; ++i){
  for (j = startpoint; j<VOCSIZE - startpoint; ++j)
   printf("%.3lf ", poolout[i][j]);
  printf("\n");
 }
 printf("\n");
}

/**********************/
/*  pool() 函数        */
/* 池化计算            */
/**********************/
void pool(double convout[][VOCSIZE]
          , double poolout[][VOCSIZE])
{
 int i, j;                                          /* 循环控制用 */
 int startpoint=FILTERSIZE/2+POOLSIZE/2;            /* 池化计算范围的下限 */

 for (i = startpoint; i<WORDLEN - startpoint; ++i)
  for (j = startpoint; j<VOCSIZE - startpoint; ++j)
   poolout[i][j] = maxpooling(convout, i, j);
}

/**********************/
/* maxpooling() 函数   */
/* 最大值池化          */
/**********************/
double maxpooling(double convout[][VOCSIZE]
                  , int i, int j)
{
 int m, n;                  /* 循环控制用 */
 double max;                /* 最大值 */
```

```
185
186 max = convout[i + POOLSIZE / 2][j + POOLSIZE / 2];
187 for (m = i - POOLSIZE / 2; m <= i + POOLSIZE / 2; ++m)
188  for (n = j - POOLSIZE / 2; n <= j + POOLSIZE / 2; ++n)
189   if (max<convout[m][n]) max = convout[m][n];
190
191  return max;
192 }
193
194 /**********************/
195 /*  convres()函数     */
196 /* 输出卷积的结果     */
197 /**********************/
198 void convres(double convout[][VOCSIZE])
199 {
200  int i, j;                                    /*循环控制用*/
201  int startpoint = FILTERSIZE / 2;             /*输出范围的下限*/
202
203  for (i = startpoint; i<WORDLEN - 1; ++i){
204   for (j = startpoint; j<VOCSIZE - 1; ++j){
205    printf("%.3lf ", convout[i][j]);
206   }
207   printf("\n");
208  }
209  printf("\n");
210 }
211
212 /**********************/
213 /*  conv()函数        */
214 /*  计算卷积          */
215 /**********************/
216 void conv(double filter[][FILTERSIZE]
217         , double sentence[][VOCSIZE], double convout[][VOCSIZE])
218 {
219  int i = 0, j = 0;                  /*循环控制用*/
220  int startpoint = FILTERSIZE / 2;   /*卷积范围的下限*/
221
222  for (i = startpoint; i<WORDLEN - startpoint; ++i)
223   for (j = startpoint; j<VOCSIZE - startpoint; ++j)
224    convout[i][j] = calcconv(filter, sentence, i, j);
225 }
226
227 /**********************/
228 /*  calcconv()函数    */
```

```
/*  滤波器的应用       */
/***********************/
double calcconv(double filter[][FILTERSIZE]
                , double sentence[][VOCSIZE], int i, int j)
{
 int m, n;                    /* 循环控制用 */
 double sum = 0;              /* 和的值 */

 for (m = 0; m<FILTERSIZE; ++m)
  for (n = 0; n<FILTERSIZE; ++n)
   sum += sentence[i-FILTERSIZE/2+m][j-FILTERSIZE/2+n] * filter[m][n];

 return sum;
}

/************************/
/* convpool() 函数 */
/* 卷积和池化 */
/************************/
void convpool(double s[WORDLEN][VOCSIZE],
              double mfilter[FILTERNO][FILTERSIZE][FILTERSIZE],
              double se[INPUTNO + 1],
              double teacher)
{
 int i, j, k;
 int startpoint = FILTERSIZE/2 + POOLSIZE/2;        /* 池化计算范围的下限 */
 /* 用各滤波器进行卷积和池化 */
 for (i = 0; i<FILTERNO; ++i){
  double convout[WORDLEN][VOCSIZE] = {0}; /* 卷积输出 */
  double poolout[WORDLEN][VOCSIZE] = {0}; /* 输出数据 */

  /* 卷积计算 */
  conv(mfilter[i], s, convout);

  /* 输出卷积计算的结果 */
  convres(convout);

  /* 池化计算 */
  pool(convout, poolout);

  /* 输出结果 */
  poolres(poolout);

  /* 将卷积计算的结果代入为全连接部分的输入 */
```

```
for (j = startpoint; j<WORDLEN - startpoint; ++j){
 for (k = startpoint; k<VOCSIZE - startpoint; ++k)
  se[i*INPUTNO/FILTERNO+(j-startpoint)*(VOCSIZE-startpoint*2)+(k-startpoint)]
   = poolout[j][k];
 }
 } /* 代入教师数据 */
 se[i*INPUTNO / FILTERNO] = teacher;

}

/**********************/
/* hlearn() 函数 */
/* 学习中间层权重 */
/**********************/
void hlearn(double wh[HIDDENNO][INPUTNO + 1]
    , double wo[HIDDENNO + 1]
    , double hi[], double e[INPUTNO + 1], double o)
{
 int i, j;           /* 循环控制用 */
 double dj;          /* 计算中间层权重用 */

 for (j = 0; j<HIDDENNO; ++j){              /* 中间层各神经元 j 作为对象 */
  dj = hi[j] * (1 - hi[j])*wo[j] * (e[INPUTNO] - o)*o*(1 - o);
  for (i = 0; i<INPUTNO; ++i)               /* 处理第 i 个权重 */
   wh[j][i] += ALPHA*e[i] * dj;
  wh[j][i] += ALPHA*(-1.0)*dj;              /* 学习阈值 */
 }
}

/**********************/
/*  getdata() 函数     */
/*  读入学习数据       */
/**********************/
int getdata(double sentence[MAXINPUTNO][WORDLEN][VOCSIZE],
            double teacher[MAXINPUTNO])
{
 int i = 0, j = 0, k = 0;                   /* 循环控制用 */

 /* 输入数据 */
 while (scanf("%lf", &teacher[i]) != EOF){ /* 读入教师数据 */
  /* 读入单词序列数据 */
  while (scanf("%lf", &sentence[i][j][k]) != EOF){
   ++k;
   if (k >= VOCSIZE){                       /* 下一个数据 */
```

```
317      k = 0;
318      ++j;
319      if (j >= WORDLEN) break;          /* 输入终止 */
320     }
321    }
322    j = 0; k = 0;
323    ++i;
324    if (i>MAXINPUTNO) break;           /* 输入终止了 */
325   }
326   return i;
327 }
328
329 /**********************/
330 /*  olearn() 函数      */
331 /*  学习输出层权重     */
332 /**********************/
333 void olearn(double wo[HIDDENNO + 1]
334     , double hi[], double e[INPUTNO + 1], double o)
335 {
336   int i;                /* 循环控制用 */
337   double d;             /* 用于计算权重 */
338
339   d = (e[INPUTNO] - o)*o*(1 - o);    /* 计算误差 */
340   for (i = 0; i<HIDDENNO; ++i)
341     wo[i] += ALPHA*hi[i] * d;        /* 学习权重 */
342   wo[i] += ALPHA*(-1.0)*d;           /* 学习阈值 */
343 }
344
345 /**********************/
346 /*  forward() 函数     */
347 /*  前向计算           */
348 /**********************/
349 double forward(double wh[HIDDENNO][INPUTNO + 1]
350   ,double wo[HIDDENNO + 1], double hi[], double e[INPUTNO + 1])
351 {
352   int i, j;             /* 循环控制用 */
353   double u;             /* 计算加权和 */
354   double o;             /* 计算输出 */
355
356   /*hi 的计算 */
357   for (i = 0; i<HIDDENNO; ++i){
358     u = 0;/* 求得加权和 */
359     for (j = 0; j<INPUTNO; ++j)
360       u += e[j] * wh[i][j];
```

```
 u -= wh[i][j];      /*处理阈值*/
 hi[i] = f(u);
}
/*计算输出o*/
o = 0;
for (i = 0; i<HIDDENNO; ++i)
 o += hi[i] * wo[i];
o -= wo[i];          /*处理阈值*/

 return f(o);
}

/**********************/
/*   print()函数      */
/*   输出结果         */
/**********************/
void print(double wh[HIDDENNO][INPUTNO + 1]
          , double wo[HIDDENNO + 1])
{
 int i, j;/*循环控制用*/

 for (i = 0; i<HIDDENNO; ++i)
  for (j = 0; j<INPUTNO + 1; ++j)
    printf("%lf ", wh[i][j]);
 printf("\n");
 for (i = 0; i<HIDDENNO + 1; ++i)
  printf("%lf ", wo[i]);
 printf("\n");
}

/**********************/
/*    initwh()函数    */
/*中间层权重的初始化   */
/**********************/
void initwh(double wh[HIDDENNO][INPUTNO + 1])
{
 int i, j;/*循环控制用*/

 /*由随机数生成权重*/
 for (i = 0; i<HIDDENNO; ++i)
  for (j = 0; j<INPUTNO + 1; ++j)
   wh[i][j] = drnd();
}

```

```
/**********************/
/*     initwo() 函数     */
/* 输出层权重的初始化      */
/**********************/
void initwo(double wo[HIDDENNO + 1])
{
 int i;/* 循环控制用 */

 /* 由随机数生成权重 */
 for (i = 0; i<HIDDENNO + 1; ++i)
  wo[i] = drnd();
}

/*******************/
/* drnd() 函数       */
/* 生成随机数         */
/*******************/
double drnd(void)
{
 double rndno;/* 生成的随机数 */

 while ((rndno = (double)rand() / RAND_MAX) == 1.0);
 rndno = rndno * 2 - 1;/* 生成 -1 和 1 之间的随机数 */
 return rndno;
}

/*******************/
/* f() 函数          */
/* 传递函数          */
/*(sigmoid 函数)    */
/*******************/
double f(double u)
{
 return 1.0 / (1.0 + exp(-u));
}
```

实例 3.4 给出了 cnn.c 程序的执行示例。该例中，给定 1-of-*N* 表示的单词序列，要学习这一单词序列出现在文章的哪个部分。有了这样的学习结果，如果给出适当的单词序列，CNN 就能够判断该单词序列是在文章的开头、中间出现，还是在末尾出现。

将学习数据集存放在 cnndata.txt 文件中。cnndada.txt 文件含有多个学习数据，每

个学习数据都由一个教师数据和与之对应的 1-of-*N* 表示的单词序列组成。在该例中，cnndada.txt 文件中有 3 个学习数据，各自的教师数据值为 0、0.5 和 1。在这些教师数据值中，0 表示文章的开头，1 表示文章的结尾，以说明 1-of-*N* 表示的单词序列在文章中出现的位置。本例中，第一个学习数据出现在文章的开头，第二个学习数据出现在文章的中间，第三个学习数据出现在文章的末尾部分。

在实例 3.4 中，以 cnndata.txt 作为学习数据集来执行 cnn.c 程序。cnn.c 程序会给出连接权重的初始值和学习数据。随着学习的进行，误差值会发生变化，cnn.c 程序会给出每次学习后得到的误差值。当误差降至预设的符号常数 LIMIT 值以下时，学习的迭代过程将终止。此后，输出通过学习所获得的网络权重。

在学习结束后，cnn.c 程序使用所得到的 CNN 对学习数据集进行计算，得出其对应的网络输出值并加以表示。在本例中，和教师数据 0 对应的网络输出值是 0.001097，和 0.5 对应的网络输出值是 0.577789，而和 1 对应的网络输出值是 0.979560。

实例 3.4 cnn.c 程序的执行示例[⊖]。

```
C:\Users\odaka\ch3>type cnndata.txt
0
1 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0 0 0 0 0
0 0 0 0 0 0 1 0 0 0 0 0
0.5
0 0 0 0 1 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0 0 0 0 0
0 0 0 0 0 0 1 0 0 0 0 0
0 0 0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0 1 0 0 0
1 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0 0 0
```

cnndata.txt 记录了 1-of-*N* 表示的单词序列、表示该单词序列在文章中哪一部分出现的数值（教师数据）

教师数据“0”表示第 1 个学习数据出现在文章的开头

学习数据 1

教师数据“0.5”表示第 2 个学习数据出现在文章的中间

学习数据 2

⊖ 译者在 Microsoft Visual Studio Professional 2013 下编译执行该程序，和原书结果略有不同（原书迭代 11905 次收敛）。——译者注

```
1
1 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0 1 0 0 0
```

教师数据“1”表示第3个学习数据出现在文章的末尾

学习数据3

通过cnn.c程序学习

```
C:\Users\odaka\ch3>cnn < cnndata.txt
-0.996277  0.063387 -0.071322 -0.570299 -0.052644 -0.602283
 0.840388 -0.280129  0.692373  0.241737 -0.705924  0.734001  0.528489
 0.086520 -0.264992 -0.620655 -0.996155  0.476424 -0.034883
-0.053194 -0.842524 -0.439375  0.736808 -0.421857  0.365642
-0.194433 -0.133213  0.175939 -0.136937 -0.320475  0.431501  0.066622
-0.996704 -0.877132 -0.392987 -0.269631 -0.680532 -0.384625
-0.283364  0.909726  0.448225  0.654286  0.046968 -0.590381  0.979736
-0.545457  0.997070  0.901364  0.295633 -0.315348 -0.878842  0.042817
 0.458174  0.049471  0.629627  0.228004  0.212134  0.534165 -0.092013
 0.277627  0.163244 -0.880184  0.619312 -0.616688 -0.806146  0.785089
-0.050020  0.619373 -0.819514 -0.395550 -0.436323  0.331767
 0.074007 -0.241981 -0.176061 -0.279885  0.405377  0.262673  0.849055
-0.077364 -0.719413 -0.761101  0.961669  0.917783  0.925718  0.525803
 0.095065 -0.445357 -0.155248  0.455184  0.719657 -0.974975 -0.345134
-0.400922 -0.149510 -0.910642 -0.342021 -0.697623
 0.700186 -0.631397 0.214637
学习数据的个数: 3
3.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 3.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 3.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 3.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 3.000 0.000 0.000 0.000 0.000 0.000

3.000 3.000 3.000 0.000 0.000 0.000 0.000 0.000
3.000 3.000 3.000 3.000 0.000 0.000 0.000 0.000
3.000 3.000 3.000 3.000 3.000 0.000 0.000 0.000
(下面继续显示和输入数据相关的信息)
1       0.629662
2       1.168984
3       1.276232
4       1.180778
5       1.210188
6       1.148700
```

随着学习的进行，误差值发生变化

```
 7       1.108045
 8       1.174452
 9       1.259530
10       1.176820
11       1.243292
12       1.169863
13       1.276409
14       1.181174
15       1.207067
(下面显示随着全连接型神经网络的学习的误差值的变化)
11932    0.137422
11933    0.026195
11934    0.009686

-0.851711 -0.127071  0.009558  0.036063  0.553718  0.004079  1.781774  0.389918  0.501915
 0.051279 -0.896382  0.814881  1.134851  0.692882  0.676394  0.565309 -1.521637 -0.049058
-0.225341 -0.243652 -0.761644  0.166987  1.678194  0.764106  0.525506 -0.034570
 0.117096  0.426249  0.113373 -0.230029  0.521947  0.492092 -0.746395 -0.717269 -0.233124
-0.019322 -0.430223 -0.469339 -0.192918  1.335196  0.363511  0.569572  0.206831
-0.430517  1.230045 -0.630172  1.247380  1.326834  0.045323 -0.445907 -0.728132
 0.755619  0.738999  0.330295  0.910452  0.227560 -0.350402  0.684875  0.058696  0.428336
 0.876045 -0.599360  0.900137 -0.617131 -1.275223  1.217066  0.381958  0.770082 -0.668804
 0.317252 -0.155499  0.331323 -0.395069 -0.379257 -0.313337 -0.229797  0.455466
 0.312761  1.036420  0.110000 -0.813317 -0.711013  0.824393  0.780507  0.975806  0.575892
 0.426422 -0.257992 -0.249152  0.786540  1.051014 -1.112251 -0.482410 -0.350833
 0.181847 -0.860553 -0.435925 -0.747711  7.128965 -3.678673  3.136625

0
3.000000 3.000000 3.000000 0.000000 0.000000 0.000000 0.000000 0.000000 3.000000
3.000000 3.000000 3.000000 0.000000 0.000000 0.000000 0.000000 3.000000 3.000000
3.000000 3.000000 3.000000 0.000000 0.000000 0.000000 1.000000 1.000000 1.000000
1.000000 1.000000 0.000000 0.000000 0.000000 1.000000 1.000000 1.000000 1.000000
1.000000 1.000000 0.000000 0.000000 1.000000 1.000000 1.000000 1.000000 1.000000
1.000000 1.000000 0.000000 0.000000
0.001097

1
0.000000 0.000000 3.000000 3.000000 3.000000 3.000000 3.000000 0.000000 0.000000
0.000000 0.000000 3.000000 3.000000 3.000000 3.000000 2.000000 0.000000 0.000000
0.000000 0.000000 3.000000 3.000000 3.000000 2.000000 0.000000 0.000000 1.000000
1.000000 1.000000 1.000000 1.000000 1.000000 1.000000 0.000000 0.000000 1.000000
1.000000 1.000000 1.000000 1.000000 1.000000 1.000000 0.000000 0.000000 1.000000
1.000000 1.000000 1.000000 0.500000
0.577789
```

学习结束，输出作为学习结果的神经网络权重

教师数据为 0，学习后的网络输出值是 0.001097

教师数据为 0.5，学习后的网络输出值是 0.577789

```
2
2.000000 1.000000 1.000000 1.000000 1.000000 1.000000 2.000000 2.000000 1.000000
1.000000 1.000000 1.000000 1.000000 1.000000 2.000000 3.000000 0.000000 0.000000
1.000000 1.000000 1.000000 1.000000 2.000000 3.000000 1.000000 1.000000 1.000000
1.000000 1.000000 0.000000 0.000000 1.000000 1.000000 1.000000 1.000000 1.000000
1.000000 0.000000 0.000000 1.000000 0.000000 0.000000 1.000000 1.000000 1.000000
0.000000 1.000000 1.000000 1.000000
0.979560
```

教师数据为 1，学习后的网络输出值是 0.979560

```
C:\Users\odaka\ch3>
```

3.4.3 基于 CNN 的单词序列评估

本节将介绍如何利用 cnn.c 程序的学习结果来对未知数据进行评估。

举例来说，考虑判断某段文本出现在文章哪个部分的评估方法。利用实例 3.4 所示的学习结果，可以评估某单词序列是出现在文章的开头部分还是在随后的部分出现。这里首先生成若干在学习数据集中未出现过的单词序列，然后应用 CNN 来评估其出现的顺序。

可以用第 2 章所给出的 makenewvec.c 程序来生成未知的单词序列。虽然 makenewvec.c 程序能够生成新文本，但它不能决定所生成的多个文本的排列顺序。这里使用 CNN 来决定所生成的多个文本间的排列顺序（图 3.17）。

为了应用 cnn.c 程序的学习结果，需要有能够将 cnn.c 程序获得的学习结果应用于未知数据的程序。下面创建的 calccnn.c 程序运用了 cnn.c 程序的学习结果，不需要进行学习，就可以求得 CNN 的输出。

calccnn.c 程序运用了 cnn.c 程序学习结果中的连接权重数据，该程序能根据输入数据计算网络输出。calccnn.c 程序和 cnn.c 程序的差别如下：

①删除了 main() 函数中的学习处理以及与学习相关的函数。

②添加了读取机制，用于读取 cnn.c 程序输出的连接权重。

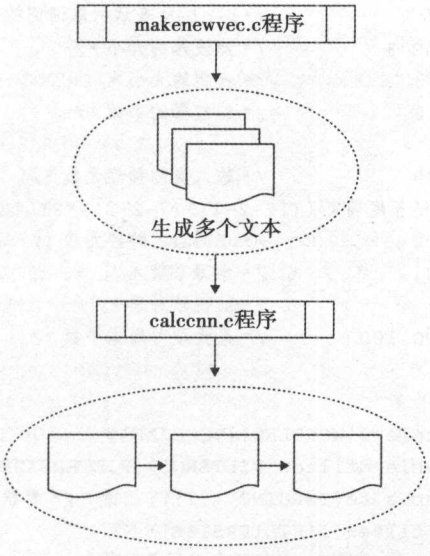


图 3.17 通过 CNN 来决定由 makenewvec.c 程序生成的文本的排列顺序

做了这样的添加和删除后，和 cnn.c 程序相比，calccnn.c 程序变得短小了。清单 3.4 给出了 calccnn.c 程序。

清单 3.4 calccnn.c 程序。

```
/************************************************************/
/*                    calccnn.c                             */
/*        利用 cnn.c 程序的学习结果计算 CNN                   */
/* 使用方法                                                 */
/* \Users\odaka\ch3>calccnn < data.txt                      */
/************************************************************/

/* 和 Visual Studio 的互换性保证 */
#define _CRT_SECURE_NO_WARNINGS

/*include 头文件 */
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

/* 符号常数的定义 */
#define VOCSIZE 12  /*1-of-N 表示的词汇数（次数）*/
```

```
#define WORDLEN 7                    /*1-of-N表示的单词词链长度 */
#define FILTERSIZE 3                 /* 滤波器的大小 */
#define POOLSIZE 3                   /* 池化的大小 */
#define FILTERNO 2                   /* 滤波器的个数 */

#define INPUTNO 48                   /* 输入层的神经元数 */
 /* 由词汇数和单词词链长度确定((12-2-2)*(7-2-2))*FILTERNO*/
#define HIDDENNO 2                   /* 中间层的神经元数 */
#define ALPHA  10                    /* 学习系数 */
#define SEED 7                       /* 随机数种子 */
#define MAXINPUTNO 100               /* 最大学习数据个数 */

/* 函数原型的声明 */
void convpool(double s[WORDLEN][VOCSIZE],
              double mfilter[FILTERNO][FILTERSIZE][FILTERSIZE],
              double se[INPUTNO + 1]);       /* 卷积和池化 */
void conv(double filter[][FILTERSIZE]
   ,double sentence[][VOCSIZE]
   ,double convcut[][VOCSIZE]);              /* 卷积计算 */
double calcconv(double filter[][FILTERSIZE]
              , double sentence[][VOCSIZE], int i, int j);
                                             /* 应用滤波器 */
void convres(double convout[][VOCSIZE]);     /* 输出卷积结果 */
int getdata(double sentence[MAXINPUTNO][WORDLEN][VOCSIZE]);
                                             /* 读入数据 */
void poolres(double poolout[][VOCSIZE]);     /* 输出池化结果 */
void pool(double convout[][VOCSIZE]
        , double poolout[][VOCSIZE]);        /* 池化计算 */
double maxpooling(double convout[][VOCSIZE]
                , int i, int j);             /* 最大值池化 */
double f(double u);                          /* 传递函数(sigmoid函数)*/
void readwh(double wh[HIDDENNO][INPUTNO + 1]);
                                             /* 中间层权重的初始化 */
void readwo(double wo[HIDDENNO + 1]);        /* 输出层权重的初始化 */

void print(double wh[HIDDENNO][INPUTNO + 1]
          ,double wo[HIDDENNO + 1]);         /* 输出结果 */
double forward(double wh[HIDDENNO][INPUTNO + 1]
         ,double wo[HIDDENNO + 1], double hi[]
         ,double e[INPUTNO + 1]);            /* 前向计算 */

/*******************/
/*    main()函数    */
/*******************/
```

```
int main()
{
 double mfilter[FILTERNO][FILTERSIZE][FILTERSIZE]
   = {
    {{1,0,0}, {0,1,0}, {0,0,1}},
    {{1,0,0}, {1,0,0}, {1,0,0}}
   };                                                    /* フィルタ群 */
double sentence[MAXINPUTNO][WORDLEN][VOCSIZE];         /* 输入数据 */
double convout[WORDLEN][VOCSIZE] = { 0 };              /* 输出卷积结果 */
double poolout[WORDLEN][VOCSIZE] = { 0 };              /* 输出数据 */

double wh[HIDDENNO][INPUTNO + 1];                  /* 中间层权重 */
double wo[HIDDENNO + 1];                           /* 输出层权重 */
double e[MAXINPUTNO][INPUTNO + 1];                 /* 学习数据集 */
double hi[HIDDENNO + 1];                           /* 中间层权重 */
double o;              /* 输出 */
int i, j;              /* 循环控制用 */
int n_of_e;            /* 学习数据的个数 */
int count = 0;         /* 重复次数的计数 */

                       /* 读入权重 */
readwh(wh);            /* 读入中间层权重 */
readwo(wo);            /* 读入输出层权重 */
print(wh, wo);         /* 输出结果 */

                       /* 读入测试数据 */
n_of_e = getdata(sentence);
printf("検査データの個数 :%d\n", n_of_e);

                       /* 卷积和池化计算 */
for (i = 0; i<n_of_e; ++i){
 convpool(sentence[i], mfilter, e[i]);
}

                       /* 输出连接权重 */
print(wh, wo);

                       /* 对于学习数据的计算输出 */
for (i = 0; i<n_of_e; ++i){
 printf("%d\n", i);
 for (j = 0; j<INPUTNO; ++j)
  printf("%lf ", e[i][j]);
 printf("\n");
 o = forward(wh, wo, hi, e[i]);
```

```
106  printf("%lf\n\n", o);
107 }
108
109 return 0;
110 }
111 /**********************/
112 /*  poolres() 函数      */
113 /*   输出结果           */
114 /**********************/
115 void poolres(double poolout[][VOCSIZE])
116 {
117  int i, j;/* 循环控制用 */
118  int startpoint = FILTERSIZE/2 + POOLSIZE / 2; /* 池化计算范围的下限 */
119
120  for (i = startpoint; i<WORDLEN - startpoint; ++i){
121   for (j = startpoint; j<VOCSIZE - startpoint; ++j)
122    printf("%.3lf ", poolout[i][j]);
123   printf("\n");
124  }
125  printf("\n");
126 }
127
128 /**********************/
129 /*  pool() 函数        */
130 /* 池化计算            */
131 /**********************/
132 void pool(double convout[][VOCSIZE]
133         , double poolout[][VOCSIZE])
134 {
135  int i, j;                                          /* 循环控制用 */
136  int startpoint = FILTERSIZE / 2 + POOLSIZE / 2;    /* 池化计算范围的下限 */
137
138  for (i = startpoint; i<WORDLEN - startpoint; ++i)
139   for (j = startpoint; j<VOCSIZE - startpoint; ++j)
140    poolout[i][j] = maxpooling(convout, i, j);
141 }
142
143 /**********************/
144 /* maxpooling() 函数   */
145 /* 最大值池化          */
146 /**********************/
147 double maxpooling(double convout[][VOCSIZE]
148                   ,int i, int j)
149 {
```

```
150  int m, n;            /* 循环控制用 */
151  double max;  /* 最大值 */
152
153  max = convout[i + POOLSIZE / 2][j + POOLSIZE / 2];
154  for (m = i - POOLSIZE / 2; m <= i + POOLSIZE / 2; ++m)
155   for (n = j - POOLSIZE / 2; n <= j + POOLSIZE / 2; ++n)
156    if (max<convout[m][n]) max = convout[m][n];
157
158  return max;
159 }
160
161
162 /**********************/
163 /*  convres() 函数      */
164 /* 输出卷积结果          */
165 /**********************/
166 void convres(double convout[][VOCSIZE])
167 {
168  int i, j;                          /* 循环控制用 */
169  int startpoint = FILTERSIZE/2;     /* 输出范围的下限 */
170
171  for (i = startpoint; i<WORDLEN - 1; ++i){
172   for (j = startpoint; j<VOCSIZE - 1; ++j){
173    printf("%.3lf ", convout[i][j]);
174   }
175   printf("\n");
176  }
177 printf("\n");
178 }
179
180 /**********************/
181 /*  conv() 函数         */
182 /*  计算卷积            */
183 /**********************/
184 void conv(double filter[][FILTERSIZE]
185          ,double sentence[][VOCSIZE], double convout[][VOCSIZE])
186 {
187  int i = 0, j = 0;                  /* 循环控制用 */
188  int startpoint = FILTERSIZE/2;     /* 卷积范围的下限 */
189
190  for (i = startpoint; i<WORDLEN - startpoint; ++i)
191  for (j = startpoint; j<VOCSIZE - startpoint; ++j)
192   convout[i][j] = calcconv(filter, sentence, i, j);
193 }
```

```
194
195 /**********************/
196 /*  calcconv() 函数     */
197 /*  应用滤波器          */
198 /**********************/
199 double calcconv(double filter[][FILTERSIZE]
200                 , double sentence[][VOCSIZE], int i, int j)
201 {
202  int m, n;            /* 循环控制用 */
203  double sum = 0;      /* 和的值 */
204
205  for (m = 0; m<FILTERSIZE; ++m)
206   for (n = 0; n<FILTERSIZE; ++n)
207    sum+=sentence[i-FILTERSIZE/2+m][j-FILTERSIZE/2+n]*filter[m][n];
208
209  return sum;
210 }
211
212 /*************************/
213 /*  convpool() 函数       */
214 /*  卷积和池化            */
215 /*************************/
216 void convpool(double s[WORDLEN][VOCSIZE],
217               double mfilter[FILTERNO][FILTERSIZE][FILTERSIZE],
218               double se[INPUTNO + 1])
219 {
220  int i, j, k;
221  int startpoint = FILTERSIZE/2 + POOLSIZE/2;      /* 池化计算范围的下限 */
222  /* 用各滤波器进行卷积和池化 */
223  for (i = 0; i<FILTERNO; ++i){
224   double convout[WORDLEN][VOCSIZE] = { 0 };       /* 卷积输出 */
225   double poolout[WORDLEN][VOCSIZE] = { 0 };       /* 输出数据 */
226   /* 计算卷积 */
227   conv(mfilter[i], s, convout);
228
229   /* 输出卷积计算的结果 */
230   convres(convout);
231
232  /* 池化计算 */
233  pool(convout, poolout);
234
235  /* 输出结果 */
236  poolres(poolout);
237
```

```
238  /* 卷积计算的结果作为全连接部的输入而代入 */
239  for (j = startpoint; j<WORDLEN - startpoint; ++j){
240   for (k = startpoint; k<VOCSIZE - startpoint; ++k)
241    se[i*INPUTNO/FILTERNO+(j-startpoint)*(VOCSIZE-startpoint*2)+(k-startpoint)]
242     = poolout[j][k];
243   }
244  }
245 }
246
247 /***********************/
248 /*  getdata() 函数      */
249 /*   读入学习数据       */
250 /***********************/
251 int getdata(double sentence[MAXINPUTNO][WORDLEN][VOCSIZE])
252 {
253  int i = 0, j = 0, k = 0;/* 循环控制用 */
254
255  /* 输入数据 */
256  while (scanf("%lf", &sentence[i][j][k]) != EOF){
257   ++k;
258   if (k >= VOCSIZE){          /* 下一个数据 */
259    k = 0;
260    ++j;
261    if (j >= WORDLEN){         /* 下一个数据集 */
262     j = 0;
263     ++i;
264    }
265   }
266   if (i>MAXINPUTNO) break;  /* 输入结束 */
267  }
268  return i;
269 }
270
271 /***********************/
272 /*  forward() 函数      */
273 /*  前向计算            */
274 /***********************/
275 double forward(double wh[HIDDENNO][INPUTNO + 1]
276  ,double wo[HIDDENNO + 1], double hi[], double e[INPUTNO + 1])
277 {
278  int i, j;              /* 循环控制用 */
279  double u;              /* 计算加权和 */
280  double o;              /* 计算输出 */
281
```

```
282 /*hi 的计算 */
283 for (i = 0; i<HIDDENNO; ++i){
284  u = 0;                  /* 求得加权和 */
285  for (j = 0; j<INPUTNO; ++j)
286   u += e[j] * wh[i][j];
287  u -= wh[i][j];          /* 处理阈值 */
288  hi[i] = f(u);
289 }
290 /* 计算输出 o*/
291 o = 0;
292 for (i = 0; i<HIDDENNO; ++i)
293  o += hi[i] * wo[i];
294 o -= wo[i];              /* 处理阈值 */
295
296 return f(o);
297 }
298
299 /**********************/
300 /*   print() 函数      */
301 /*   输出结果          */
302 /**********************/
303 void print(double wh[HIDDENNO][INPUTNO + 1]
304           ,double wo[HIDDENNO + 1])
305 {
306  int i, j;                    /* 循环控制用 */
307
308  for (i = 0; i<HIDDENNO; ++i)
309   for (j = 0; j<INPUTNO + 1; ++j)
310    printf("%lf ", wh[i][j]);
311   printf("\n");
312   for (i = 0; i<HIDDENNO + 1; ++i)
313    printf("%lf ", wo[i]);
314   printf("\n");
315 }
316
317 /************************/
318 /*    readwh() 函数      */
319 /*    读入中间层权重     */
320 /************************/
321 void readwh(double wh[HIDDENNO][INPUTNO + 1])
322 {
323  int i, j;               /* 循环控制用 */
324
325  /* 读入权重 */
```

```
326  for (i = 0; i<HIDDENNO; ++i)
327   for (j = 0; j<INPUTNO + 1; ++j)
328    scanf("%lf", &(wh[i][j]));
329 }
330
331 /************************/
332 /*     readwo() 函数       */
333 /*     读入输出层权重       */
334 /************************/
335 void readwo(double wo[HIDDENNO + 1])
336 {
337  int i;                /* 循环控制用 */
338
339  /* 读入权重 */
340  for (i = 0; i<HIDDENNO + 1; ++i)
341   scanf("%lf", &(wo[i]));
342 }
343
344 /*******************/
345 /* f() 函数          */
346 /* 传递函数          */
347 /*(sigmoid 函数)     */
348 /*******************/
349 double f(double u)
350 {
351  return 1.0/(1.0+exp(-u));
352 }
```

为了运用 calccnn.c 程序来进行计算，需要预先使用 cnn.c 程序来训练神经网络。正如实例 3.4 所示，在 cnn.c 学习结束后，输出网络权重作为其学习结果，calccnn.c 程序就可以使用这些权重进行计算。calccnn.c 程序的输入数据包含两部分，一部分是权重，另一部分是作为计算对象的 1-of-*N* 表示的单词序列。为了能执行 calccnn.c 程序，需要先准备好上述数据（图 3.18）。

实例 3.5～实例 3.7 给出了应用 calccnn.c 程序确定 makenewvec.c 程序所生成文本的排列顺序的执行示例。

在实例 3.5 中，首先将 text3.txt 中存放的文本用 makew1gram.c 程序变换成单词 1-gram，将变换结果存放在 w1gram.txt 文件中。然后，用 makevec.c 程序将单词 1-gram

表示变换为 1-of-*N* 表示，此时可知原始文本中含有的单词数是 45。最后，在此结果基础上，采用 3 个不同的起始符编号，执行 3 次 makenewvec.c 程序，生成 3 个新文本（s1.txt～s3.txt）。

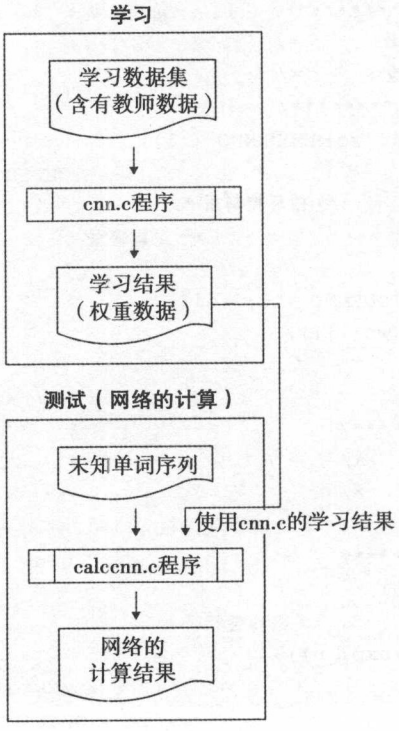


图 3.18 calccnn.c 程序的运行过程

实例 3.5 通过 calccnn.c 程序确定文本的排列顺序（1）。

```
C:\Users\odaka\ch3>type text3.txt
  自然言語処理の技術を用いると、文書の要約や、文書どうしの類似性を評価することが
できます。文書要約においては、ある文書に含まれ用語のうちから文書の特徴を表す重要
語を抽出したり、文書を表現する要約文を作成する技術が利用されています。また、こう
した技術を用いて、複数の文書どうしの類似性を数値で評価する手法が提案されています。

C:\Users\odaka\ch3>makew1gram < text3.txt > w1gram.txt

C:\Users\odaka\ch3>makevec > makevecout.txt
```

生成的原始数据文本的内容（text3.txt 文件）

用 makew1gram.c 程序将文本变换成单词的 1-gram

将单词 1-gram 表示变换为 1-of-*N* 表示

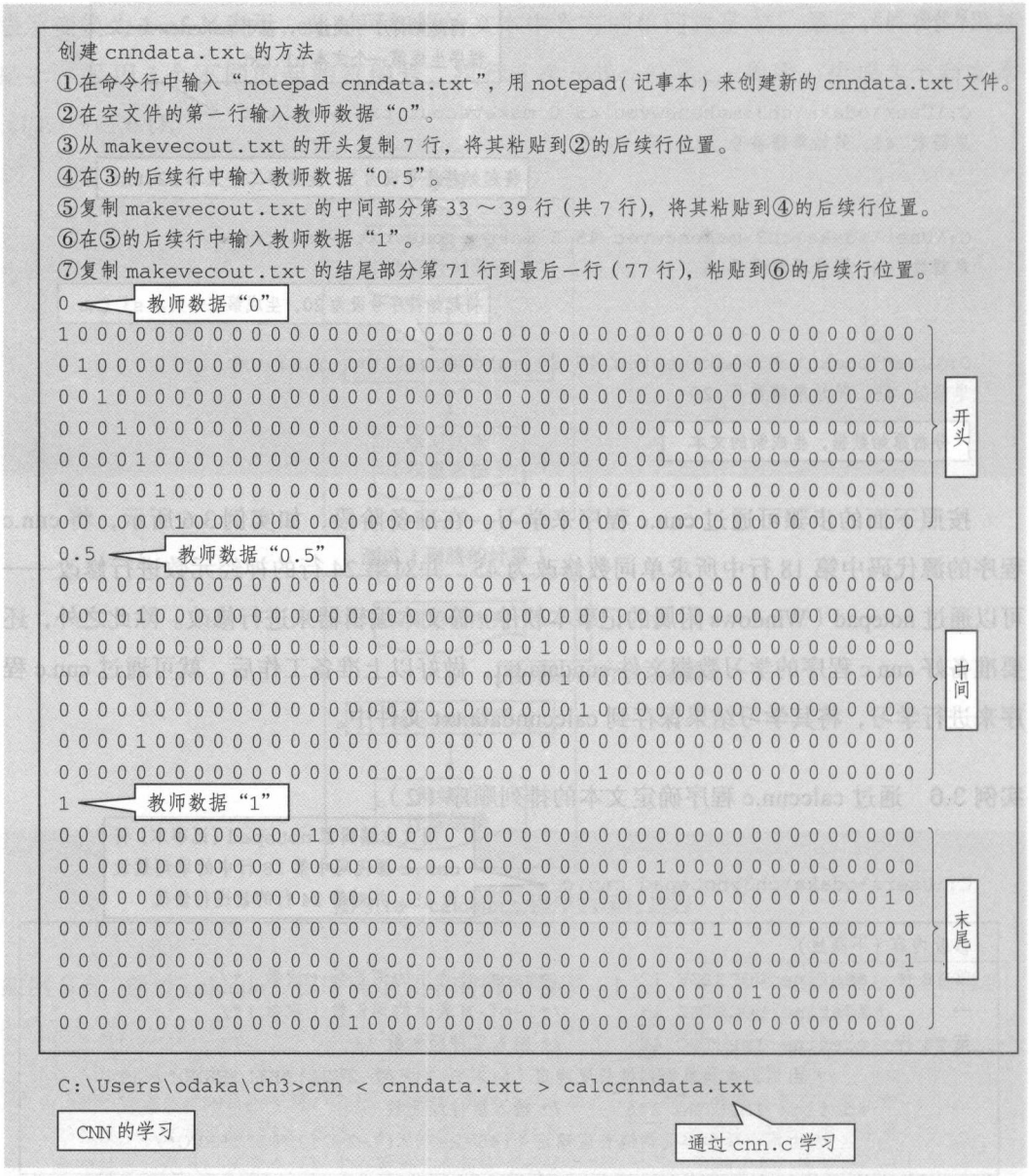
```
単語数 45
                                        将起始符序号设为 0，基于 makenewvec.c
                                        程序生成第一个文本 s1.txt
C:\User\odaka\ch3>makenewvec 45 0 makevecout.txt > s1.txt
単語数 45，開始単語番号 0
                                        将起始符序号设为 5，生成第二个文本 s2.txt
C:\User\odaka\ch3>makenewvec 45 5 makevecout.txt > s2.txt
単語数 45，開始単語番号 5
                                        将起始符序号设为 20，生成第三个文本 s3.txt
C:\User\odaka\ch3>makenewvec 45 20 makevecout.txt > s3.txt
単語数 45，開始単語番号 20
分析原始数据，生成新的文本
```

按照下面的步骤可通过 cnn.c 程序来学习。在准备阶段，如实例 3.6 所示，将 cnn.c 程序的源代码中第 18 行中所求单词数修改为 45，并对第 24 行的神经元数进行修改——可以通过 notepad（Windows 附属的记事本软件）等文本编辑器来进行修改。除此之外，还要准备好 cnn.c 程序的学习数据文件 cnndata.txt。做好以上准备工作后，就可通过 cnn.c 程序来进行学习，将其学习结果保存到 calccnndata.txt 文件中。

实例 3.6 通过 calccnn.c 程序确定文本的排列顺序（2）。

```
C:\Users\odaka\ch3>notepad cnn.c
                                        用文本编辑器 notepad（记事本）将
                                        cnn.c 源代码中第 18 行中的单词数设
                                        为 45，并对第 24 行的数进行修改
修改内容（下画线）
第 18 行  #define VOCSIZE 12        /*1-of-N 表示的词汇数（次数）*/
→         #define VOCSIZE 45        /*1-of-N 表示的词汇数（次数）*/
第 24 行  #define INPUTNO 48        /* 输入层神经元数 */
            /* 由词汇数和单词词链长度确定（(12-2-2)*(7-2-2))*FILTERNO*/
→         #define INPUTNO 246       /* 输入层神经元数 */
            /* 由词汇数和单词词链长度确定（(45-2-2)*(7-2-2))*FILTERNO*/

C:\Users\odaka\ch3>gcc cnn.c -o cnn
                                        重新编译修改后的 cnn.c
C:\Users\odaka\ch3>notepad cnndata.txt
                                        由 makevecout.txt 文件创建 cnn.c
                                        的输入数据 cnndata.txt
```



在 CNN 的学习结束后，用 calccnn.c 程序来确定 s1.txt～s3.txt 这 3 个文本的排列顺序。为此，首先要创建 calccnn.c 程序的输入数据。除此之外，还要对 calccnn.c 程序的一部分进行改写。实例 3.7 给出了这一操作的示例，和 cnn.c 一样，需要根据处理对象数据来修改其单词数。做完上述准备工作后，就可以用 calccnn.c 程序来确定文本的顺序了。

实例 3.7 通过 calccnn.c 程序确定文本的排列顺序（3）。

```
C:\Users\odaka\ch3>notepad calccnndata.txt
```

用文本编辑器 notepad（记事本）来编辑 calccnndata.txt 文件

```
变更操作
①从 cnn.c 的输出数据中提取出神经网络权重的信息。
②将要输入到 calccnn.c 程序中 1-of-N 表示的单词序列（s1.txt ~ s3.txt）添加到文件中。
  此时，每个单词序列由 7 行（7 个单词）组成。
```

```
C:\Users\odaka\ch3>type calccnndata.txt
 0.965328 0.327521 0.192812 -0.337114 0.180541 -0.602283  0.840388
-0.280129 0.692373 0.241737 -0.705924 0.600787  0.395275 -0.046694
(下面继续输出权重)
-3.713698 2.542562 -1.207277
```

编辑后 calccnndata.txt 文件的内容⊖

在 calccnndata.txt 文件中，前半部分存放的是连接权重，后半部分存放的是 1-of-N 表示的文本数据

```
1 0 0 0
```

```
1
0.000000 1.000000 1.000000 1.000000 1.000000 0.000000 0.000000
0.000000 0.000000
(中间省略)
0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
0.508779
```

学习后的 CNN 对 s2.txt 的评估结果（0.508779）

```
2
0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
0.000000 0.000000
(中间省略)

1.000000 1.000000 0.000000 0.000000 0.000000 0.000000
0.976924
```

学习后的 CNN 对 s3.txt 的评估结果（0.976924）

比较这 3 个评估值，其结果按照 s1.txt→s2.txt→s3.txt 的顺序排列

文本排列顺序的确定

通过 calcnn.c 程序对 s1.txt～s3.txt 的评估值如下所示：

- s1.txt：0.076898
- s2.txt：0.508779
- s3.txt：0.976924

从这一结果可以知道，这 3 个文本按照 s1.txt → s2.txt → s3.txt 的顺序进行排列。基于这一结果，用 makes.c 程序将 1-of-*N* 表示转换为通常的自然语言单词表示，所得的最终文本如图 3.19 所示：

自然言語処理の特徴を表す重要語を数値で評価する手法が利用されています。用いると、ある文書に含まれる用語のうちから文書を用いると、文書どうしの類似性を評価することができます。文書要約においては、こうした技術を評価することができます。

图 3.19 一连串处理后产生的文本（最终结果）

CHAPTER 4

第 4 章

文本生成与深度学习

本章介绍如何利用深度学习方法来生成文本。具体介绍所采用的循环神经网络方法，给出应用循环神经网络来生成自然语言文本的示例。

4.1 基于循环神经网络的文本生成

4.1.1 神经网络和文本生成

第 2 章介绍了文本生成的一种具体方法。在该方法中，对于单词 2-gram 按概率进行选择，将单词组合起来生成文本。除此之外，还有多种利用单词 2-gram 生成文本的方法。

有一种方法是使用神经网络来求得单词的排列顺序。将单词通过 1-of-N 表示来记录，提供给图 4.1 所示的层次型神经网络。在第 3 章中，将层次型神经网络的输出设为一个数值，而在图 4.1 中，输出层采用了和输入层相同个数的人工神经元，可以并列输出多个数值。这样构成的神经网络，接收一个 1-of-N 表示的单词输入，就可以输出对应的 1-of-N 表示的单词。

对于图 4.1 所示的神经网络，可采用适当的单词词链进行学习。如图 4.2 所示，将连续的单词组，也就是单词 2-gram，作为该神经网络的学习数据。训练神经网络，使得输

入某单词时该网络输出其后续单词。

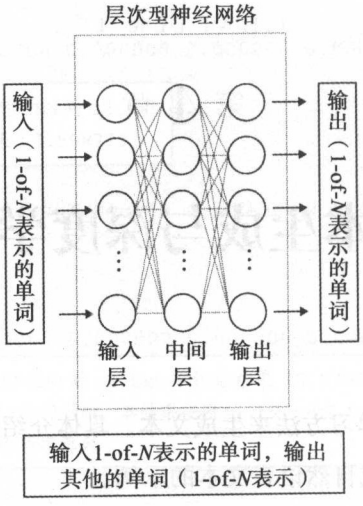


图 4.1 层次型神经网络

学习数据集（单词 2-gram 的集合）

单词 2-gram 的序号	输入数据	教师数据
1	最初的单词（学习数据 1）	第 2 个单词（教师数据 1）
2	第 2 个单词（学习数据 2）	第 3 个单词（教师数据 2）
3	第 3 个单词（学习数据 3）	第 4 个单词（教师数据 3）
4	第 4 个单词（学习数据 4）	第 5 个单词（教师数据 4）
.....		

图 4.2 单词 2-gram（两个连续单词）所构成的学习数据集

这样学习下去，得到的神经网络能够输出某单词的后续单词。采用这样的神经网络，从某个单词开始，就能根据单词的连续性质一个一个地生成新单词。其结果就是从某个单词开始能够生成单词序列——文本。然而，这样学习得到的神经网络往往不能给出有用的结果，这是由以下原因造成的。

首先考虑这样的情形，在学习数据集中，某单词的后续单词是唯一的。将这一学习数据集提供给神经网络进行学习，该神经网络会学得学习数据集中单词 2-gram 的连续关

系。这样一来，当给定某单词时，神经网络会输出学习数据集所含原文中该单词的后续单词。其结果就是，学习后的神经网络会从指定的单词开始从原文中切分出一部分文本并输出。该神经网络只具有这一功能，不具备其他任何功能（图 4.3）。

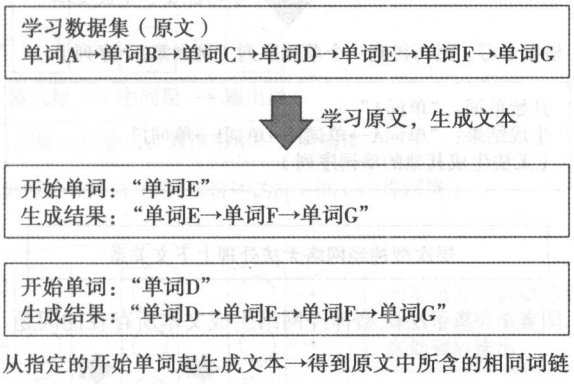


图 4.3　示例：对于原文的学习结果，神经网络只能输出原文的一部分

下面考虑在作为学习数据集的原文中，某单词的后续单词有两种以上可能的情形。此时，不仅是神经网络，无论采用哪种学习方法，从原理上讲，仅由前面的单词来决定后续单词是不可能的。如果给图 4.1 所示的层次型神经网络提供这样的学习数据集，就意味着要求给神经网络输入相同的数据而得到不同的输出结果，这本身就是一个矛盾，因而其学习就会半途而废。

针对上述情况，在学习未达到收敛时终止学习过程，将测试数据输入到神经网络中，尽管有多个输出候选者，每次测试时神经网络都会输出相同的单词而不会输出其他候选者。也就是说，神经网络只能输出其死记硬背的并不完整的原文的一部分（图 4.4）。

以上问题可以通过一句话来总结：层次型神经网络无法处理上下文（context）关系。这里所说的“上下文”是指，判断某单词的后续单词，不仅由该单词自身决定，还和该单词之前出现的单词相关。

为了解决这一问题，需要创建考虑了上下文关系的单词词链。也就是说，不仅要靠当前的输入单词来确定下一个单词，还要利用更前面的单词信息来生成单词词链。

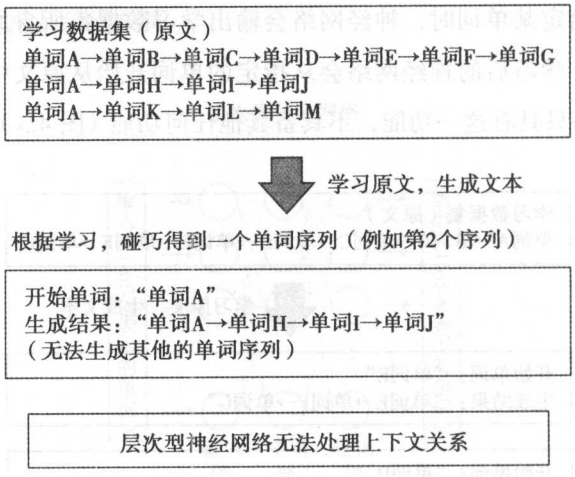


图 4.4 基于层次型神经网络生成文本所存在的问题

例如，在图 4.5 所示的示例中，选择单词“红色的”的后续单词时，可以根据“红色的”之前出现过的单词来进行选择。该例中，如果前面是“似乎很好吃的”的话，则生成“红色的苹果”会比较好。如果前面是“闪耀的”，则生成“红色的太阳”。为了能够处理上下文关系，需要构造可实现上述功能的神经网络结构。

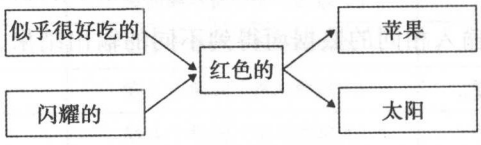


图 4.5 由于上下文不同，单词词链发生变化

事实上，不仅在应用神经网络时必须处理上下文关系，在应用其他方法时也是同样必要的。这里仅讨论使用神经网络处理上下文关系的方法。

4.1.2 循环神经网络

如上所述，为了用神经网络处理上下文关系，除了在某个时间点上给神经网络输入单词之外，也必须记住过去所输入的单词。事实上，存在能进行这种处理的神经网络，即循环神经网络（Recurrent Neural Network，RNN）。循环神经网络的概念图如图 4.6 所示。

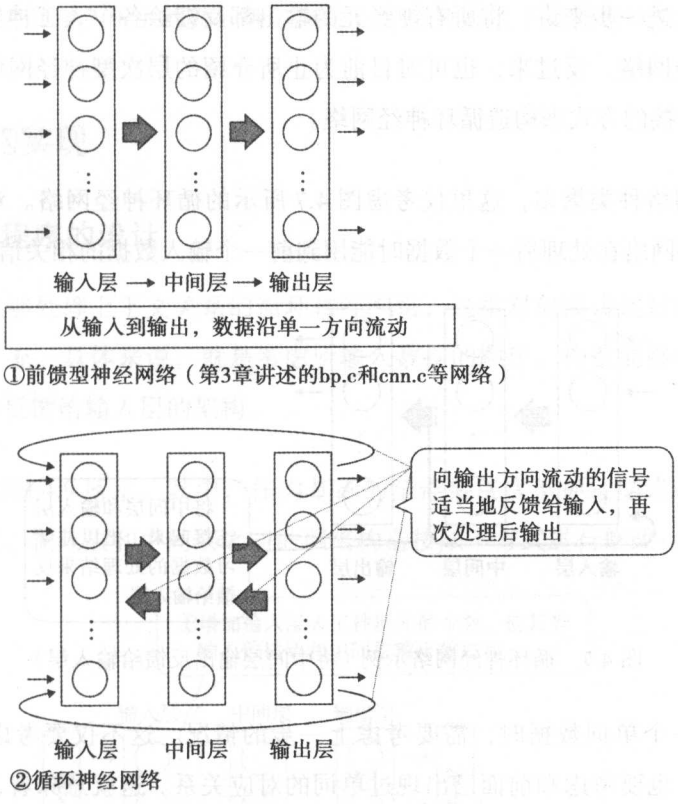


图 4.6 循环神经网络的概念图

在第 3 章讨论的层次型神经网络中，输入数据经过各层人工神经元的处理后，向输出方向沿单一方向流动前进，这样的神经网络称为前馈型神经网络。第 3 章讲述的全连接型神经网络程序 bp.c 和卷积神经网络 cnn.c 程序，所实现的都是前馈型神经网络。

与之相对应的是，在循环神经网络中，从输入到输出之间信号的流动不是单一方向的。在循环神经网络中，流向输出方向的信号会适当地反馈给输入，然后和下一个输入信号合并起来进行处理，以得到网络输出。经过这样的处理后，网络内部会记住对以往输入的处理内容，在进一步处理新输入数据时，网络将其记住的关于以往输入的处理内容和新输入的数据综合起来进行处理。

循环神经网络的形式多种多样。例如，在被称为 Hopfield 网络（Hopfield network）的循环神经网络中，任意人工神经元的输入都包括除该神经元自身的输出之外其他人工

神经元的输出。进一步来讲，将所有神经元的输出都反馈给各个人工神经元，就可以构造全连接型神经网络。反过来，也可对目前为止所介绍的层次型神经网络通过仅在有限范围内增加新连接的方式来构造循环神经网络。

循环神经网络种类繁多，这里仅考虑图 4.7 所示的循环神经网络。对于按顺序输入的数据序列，该网络在处理后一个数据时能用到前一个输入数据的相关信息。

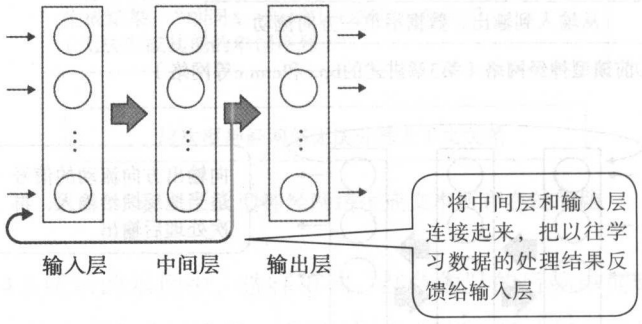


图 4.7 循环神经网络示例（将中间层输出反馈给输入层）

在处理下一个单词数据时，需要考虑上一步的情况。这不仅要考虑和当前单词的一一对应关系，也要考虑和前面已出现过单词的对应关系。这就意味着，在考虑上下文关系的情况下，可以使用循环神经网络进行单词选择（图 4.8）。

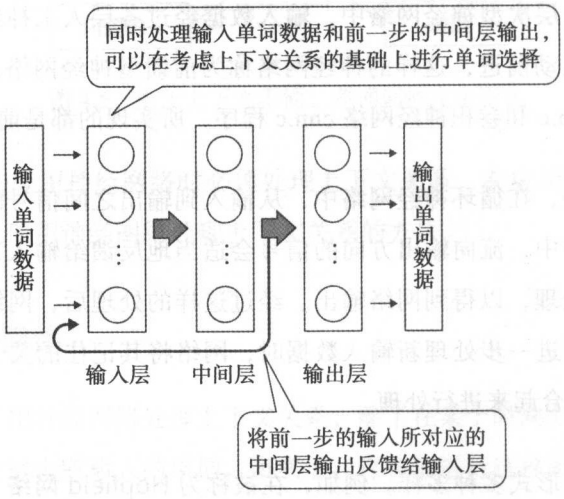


图 4.8 基于循环神经网络处理上下文关系

下一节将构造图 4.7 所示的循环神经网络，介绍考虑上下文关系后生成文本的方法。

4.2 RNN 的实现

4.2.1 RNN 程序的设计

为了构造能够处理上下文关系的循环神经网络，这里对前面讲述过的全连接层次型神经网络进行扩充。具体来说，就是考虑所输入数据的顺序，构造能够将层次型神经网络的中间层输出反馈给输入层的架构。

基本架构如图 4.9 所示。其中，作为基础的全连接层次型神经网络的输出层由多个人工神经元构成，输入层和中间层之间的网络构造按如下方式进行修改：

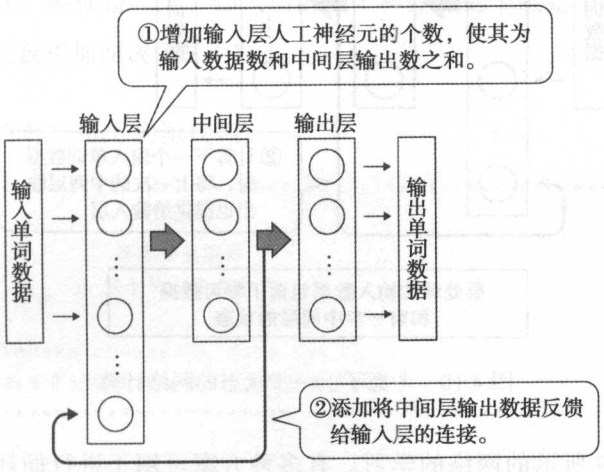


图 4.9 构造处理上下文关系的循环神经网络

①增加输入层人工神经元的个数，使其为输入数据数和中间层输出数之和。

②添加将中间层输出数据反馈给输入层的连接。

按上面两点进行扩充，就成功构造了基本的循环神经网络，这一神经网络能够处理过去已出现过的单词信息。

在图 4.9 所示的循环神经网络中，考虑由输入数据计算得到输出数据的方法。基本来说，和前馈型神经网络的情形相同，按照输入层到中间层再到输出层的顺序进行计算。但是，在提供给输入层的数据中，有一部分是由上一个数据所计算得到的中间层输出。之所以这样做，是因为需要将中间层的输出值作为记忆保存起来。这样一来，神经网络就不是单纯地根据输入数据来计算输出值，在输出值中也包含了之前输入数据的相关信息（图 4.10）。

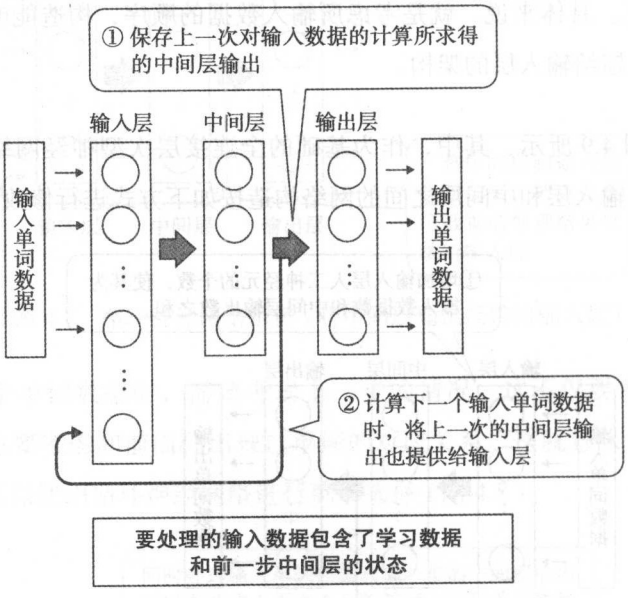


图 4.10 考虑了上一步状态的网络计算

下面考虑图 4.9 所示的网络的学习。有多种方案可用于进行循环神经网络的学习，这里采用的是前馈型神经网络的反向传播算法。也就是说，和前馈型神经网络相同，对于某个输入计算其输出，将输出值和教师数据之间的差异定义为误差，基于误差的值来对网络的参数进行调整。

实际上，循环神经网络也应该考虑由过去的输入数据导致的误差。严格地说，循环神经网络不仅要考虑学习一步之前数据的误差，也要考虑学习两步之前、三步之前乃至追溯到学习过去所有数据导致的误差。为了简单起见，这里不考虑两步之前的数据所带

来的误差，而仅基于某个时间点上网络输出的误差来进行学习。

4.2.2 RNN 程序的实现

这里基于前面所述的方案构造循环神经网络程序 rnn.c。在 rnn.c 程序中，对前一章所述的 bp.c 程序进行了修改，将上一步数据计算得到的中间层输出结果反馈到输入层。为此，输入层的人工神经元个数中，除了 1-of-*N* 表示所需的向量元素数，还包括中间层的输出数。要注意的是，在由输入数据计算输出值的过程中，一定要使用由中间层反馈的数据。

清单 4.1 给出了循环神经网络 rnn.c。在 rnn.c 程序中，将输入的 1-of-*N* 表示的向量元素数设为 5，将中间层和输出层的人工神经元数设为 5，即 rnn.c 程序是关于这一 5-5-5 神经网络的计算程序。顺便说一下，输入层的人工神经元的个数是 10——由向量的元素数和中间层的神经元数相加而成（图 4.11）。

清单 4.1 循环神经网络 rnn.c 的源代码。

```
/*************************************************************/
/*                        rnn.c                              */
/*                    循环神经网络                           */
/*                   基于反向传播学习                        */
/* 使用方法                                                  */
/*  \Users\odaka\ch4>rnn < data.txt                          */
/*  输出误差的变化过程及作为学习结果的连接系数等            */
/*************************************************************/

/* 和 Visual Studio 的互换性保证 */
#define _CRT_SECURE_NO_WARNINGS

/* include 头文件 */
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

/* 符号常数的定义 */
#define INPUTNO 5      /* 输入的元素数 */
#define HIDDENNO 5     /* 中间层神经元数 */
```

```
#define OUTPUTNO 5                  /* 输出层神经元数 */
#define ALPHA  10                   /* 学习系数 */
#define SEED 65535                  /* 随机数的种子 */
//#define SEED 7                    /* 随机数的种子（其他值）*/
#define MAXINPUTNO 100              /* 学习数据的最大个数 */
#define BIGNUM 100                  /* 误差的初始值 */
#define LIMIT 0.01                  /* 误差的上限值 */

/* 函数原型的声明 */
double f(double u) ;                /* 传递函数（sigmoid 函数）*/
void initwh(double wh[HIDDENNO][INPUTNO+1+HIDDENNO]) ;
                                    /* 中间层权重的初始化 */
void initwo(double wo[HIDDENNO+1]);          /* 输出层权重的初始化 */
double drnd(void) ;                          /* 生成随机数 */
void print(double wh[HIDDENNO][INPUTNO+1+HIDDENNO]
          ,double wo[OUTPUTNO][HIDDENNO+1]);/* 输出结果 */
double forward(double wh[HIDDENNO][INPUTNO+1+HIDDENNO]
          ,double wo[HIDDENNO+1],double hi[]
          ,double e[]) ;                     /* 前向计算 */
void olearn(double wo[HIDDENNO+1],double hi[]
          ,double e[],double o,int k) ;      /* 输出层权重的调整 */
int getdata(double e[][INPUTNO+OUTPUTNO+HIDDENNO]);
                                             /* 读入学习数据 */
void hlearn(double wh[HIDDENNO][INPUTNO+1+HIDDENNO]
          ,double wo[HIDDENNO+1],double hi[]
          ,double e[],double o,int k) ;      /* 中间层权重的调整 */

/********************/
/*     main() 函数    */
/********************/
int main()
{
 double wh[HIDDENNO][INPUTNO+1+HIDDENNO] ;          /* 中间层权重 */
 double wo[OUTPUTNO][HIDDENNO+1] ;                  /* 输出层权重 */
 double e[MAXINPUTNO][INPUTNO+OUTPUTNO+HIDDENNO] ; /* 学习数据集 */
 double hi[HIDDENNO+1]={0} ;          /* 中间层的输出 */
 double o[OUTPUTNO] ;                 /* 输出 */
 double err=BIGNUM ;                  /* 误差评估 */
 int i,j,k ;                          /* 循环控制用 */
 int n_of_e ;                         /* 学习数据的个数 */
 int count=0 ;                        /* 循环次数计数 */
 double errsum=BIGNUM ;               /* 误差的累计 */

                                      /* 初始化随机数 */
```

```
65  srand(SEED) ;
66
67  /*初始化权重*/
68  initwh(wh) ;          /*初始化中间层权重*/
69  for(i=0;i<OUTPUTNO;++i)
70   initwo(wo[i]) ;      /*初始化输出层权重*/
71
72  /*读入学习数据*/
73  n_of_e=getdata(e) ;
74  fprintf(stderr,"学習データの個数:%d\n",n_of_e) ;
75
76  /*学习*/
77  while(errsum>LIMIT){
78   /*对应多个输出层*/
79   errsum=0 ;
80   for(k=0;k<OUTPUTNO;++k){
81    err=0.0 ;
82    for(j=0;j<n_of_e;++j){
83    /*将前一层中间层的输出追加到输入中*/
84     for(i=0;i<HIDDENNO;++i)
85       e[j][INPUTNO+i]=hi[i] ;
86     /*前向计算*/
87     o[k]=forward(wh,wo[k],hi,e[j]) ;
88     /*输出层权重的调整*/
89     olearn(wo[k],hi,e[j],o[k],k) ;
90     /*中间层权重的调整*/
91     hlearn(wh,wo[k],hi,e[j],o[k],k) ;
92     /*误差的累积*/
93     err+=(o[k]-e[j][INPUTNO+k+HIDDENNO])*(o[k]-e[j][INPUTNO+k+HIDDENNO]) ;
94    }
95    ++count ;
96    /*误差的合计值*/
97    errsum+=err ;
98    /*对应多个输出层结束*/
99   }
100  /*输出误差*/
101  fprintf(stderr,"%d\t%lf\n",count,errsum) ;
102 }/*学习结束*/
103
104 /*输出连接权重*/
105 print(wh,wo) ;
106
107 /*输出学习数据的计算结果*/
108 for(i=0;i<n_of_e;++i){
```

```
109   fprintf(stderr,"%d:\n",i) ;
110   for(j=0;j<INPUTNO+HIDDENNO;++j)
111    fprintf(stderr,"%.3lf ",e[i][j]) ;
112   fprintf(stderr,"\n") ;
113   for(j=INPUTNO+HIDDENNO;j<INPUTNO+OUTPUTNO+HIDDENNO;++j)
114    fprintf(stderr,"%.3lf ",e[i][j]) ;
115   fprintf(stderr,"\n") ;
116   for(j=0;j<OUTPUTNO;++j)
117    fprintf(stderr,"%.3lf ",forward(wh,wo[j],hi,e[i])) ;
118   /* 将前一次中间层的输出追加到输入中 */
119   if(i<n_of_e-1)
120    for(j=0;j<HIDDENNO;++j)
121     e[i+1][INPUTNO+j]=hi[j] ;
122   fprintf(stderr,"\n") ;
123  }
124
125  return 0 ;
126 }
127
128 /**********************/
129 /*  hlearn() 函数       */
130 /*  中间层权重的学习     */
131 /**********************/
132 void hlearn(double wh[HIDDENNO][INPUTNO+1+HIDDENNO]
133     ,double wo[HIDDENNO+1]
134     ,double hi[],double e[],double o,int k)
135 {
136  int i,j ;            /* 循环控制用 */
137  double dj ;          /* 用于计算中间层权重 */
138
139  for(j=0;j<HIDDENNO;++j){          /* 将中间层各神经元 j 作为对象 */
140   dj=hi[j]*(1-hi[j])*wo[j]*(e[INPUTNO+k+HIDDENNO]-o)*o*(1-o) ;
141   for(i=0;i<INPUTNO+HIDDENNO;++i)  /* 处理第 i 个权重 */
142    wh[j][i]+=ALPHA*e[i]*dj ;
143   wh[j][i]+=ALPHA*(-1.0)*dj ;      /* 学习阈值 */
144  }
145 }
146
147 /**********************/
148 /*  getdata() 函数      */
149 /*   读入学习数据        */
150 /**********************/
151 int getdata(double e[][INPUTNO+OUTPUTNO+HIDDENNO])
152 {
```

```
153  int n_of_e=0 ;        /* 数据集的个数 */
154  int j=0 ;             /* 循环控制用 */
155
156  /* 输入数据 */
157  while(scanf("%lf",&e[n_of_e][j])!=EOF){
158   ++ j ;
159   if(j==INPUTNO) j+=HIDDENNO;                    /* 跳过循环成分读 */
160   if(j>=INPUTNO+OUTPUTNO+HIDDENNO){              /* 下一个数据 */
161    j=0 ;
162    ++n_of_e ;
163    if(n_of_e>=MAXINPUTNO) break ;                /* 到达数据数的上限 */
164   }
165  }
166
167  return n_of_e ;
168 }
169
170 /**********************/
171 /* olearn() 函数       */
172 /* 学习输出层权重       */
173 /**********************/
174 void olearn(double wo[HIDDENNO+1]
175     ,double hi[],double e[],double o,int k)
176 {
177  int i ;               /* 循环控制用 */
178  double d ;            /* 用于计算权重 */
179
180  d=(e[INPUTNO+k+HIDDENNO]-o)*o*(1-o) ;          /* 计算误差 */
181  for(i=0;i<HIDDENNO;++i){
182   wo[i]+=ALPHA*hi[i]*d ;                        /* 学习权重 */
183  }
184  wo[i]+=ALPHA*(-1.0)*d ;                        /* 学习阈值 */
185 }
186
187 /**********************/
188 /*  forward() 函数     */
189 /*  前向计算           */
190 /**********************/
191 double forward(double wh[HIDDENNO][INPUTNO+1+HIDDENNO]
192  ,double wo[HIDDENNO+1],double hi[],double e[])
193 {
194  int i,j ;             /* 循环控制用 */
195  double u ;            /* 计算加权和 */
196  double o ;            /* 计算输出值 */
```

```
197
198  /*hi 的计算 */
199  for(i=0;i<HIDDENNO;++i){
200   u=0 ;                /* 求得加权和 */
201   for(j=0;j<INPUTNO+HIDDENNO;++j)
202    u+=e[j]*wh[i][j] ;
203   u-=wh[i][j] ;        /* 阈值的处理 */
204   hi[i]=f(u) ;
205  }
206  /* 计算输出 o*/
207  o=0 ;
208  for(i=0;i<HIDDENNO;++i)
209   o+=hi[i]*wo[i] ;
210  o-=wo[i] ;            /* 阈值的处理 */
211
212  return f(o) ;
213 }
214
215 /***********************/
216 /*   print() 函数       */
217 /*   输出结果           */
218 /***********************/
219 void print(double wh[HIDDENNO][INPUTNO+1+HIDDENNO]
220           ,double wo[OUTPUTNO][HIDDENNO+1])
221 {
222  int i,j ;             /* 循环控制用 */
223
224  for(i=0;i<HIDDENNO;++i){
225   for(j=0;j<INPUTNO+1+HIDDENNO;++j)
226    printf("%.3lf ",wh[i][j]) ;
227   printf("\n") ;
228  }
229  printf("\n") ;
230  for(i=0;i<OUTPUTNO;++i){
231   for(j=0;j<HIDDENNO+1;++j)
232    printf("%.3lf ",wo[i][j]) ;
233   printf("\n") ;
234  }
235  printf("\n") ;
236 }
237
238 /***********************/
239 /*    initwh() 函数     */
240 /* 中间层权重的初始化    */
241 /***********************/
```

```
void initwh(double wh[HIDDENNO][INPUTNO+1+HIDDENNO])
{
 int i,j ;    /* 循环控制用 */

 /* 根据随机数确定权重 */
 for(i=0;i<HIDDENNO;++i)
  for(j=0;j<INPUTNO+1+HIDDENNO;++j)
   wh[i][j]=drnd() ;
}

/**********************/
/*    initwo() 函数    */
/* 输出层权重的初始化    */
/**********************/
void initwo(double wo[HIDDENNO+1])
{
 int i ;/* 循环控制用 */

 /* 根据随机数确定权重 */
 for(i=0;i<HIDDENNO+1;++i)
   wo[i]=drnd() ;
}

/*******************/
/* drnd() 函数      */
/* 生成随机数       */
/*******************/
double drnd(void)
{
 double rndno ;                    /* 生成的随机数 */

 while((rndno=(double)rand()/RAND_MAX)==1.0) ;
 rndno=rndno*2-1 ;                 /* 生成 -1 到 1 之间的随机数 */
 return rndno;
}

/*******************/
/* f() 函数         */
/* 传递函数         */
/* (sigmoid 函数)   */
/*******************/
double f(double u)
{
 return 1.0/(1.0+exp(-u)) ;
}
```

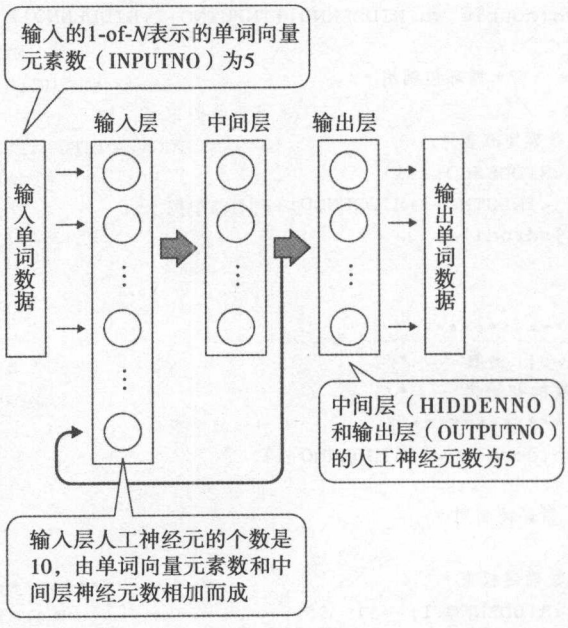


图 4.11 在循环神经网络 rnn.c 中，输入输出的关系

在 rnn.c 程序中，学习数据集采用的是 1-of-*N* 表示的单词 2-gram。读入学习数据集后，使用误差反向传播算法来对神经网络的参数进行调整。经过多次迭代后，当误差的值降至预定值之下时终止学习。学习结果既包括循环神经网络的权重和阈值，也包括学习数据集所对应的网络输出值。

在 rnn.c 程序中，当输出学习结果中网络的权重和阈值时，采用标准输出来保存输出值到文件中。也就是说，将学习结果中的这些值保存到文件中，该文件可供后面执行文本生成程序 calcrnn.c 时所用（下一节将给出关于 calcrnn.c 程序的说明）。当输出学习过程及学习数据集所对应的计算结果时，采用标准误差输出。图 4.12 给出了关于以上关系的总结。

和其他神经网络程序一样，rnn.c 程序的执行结果也依赖于运行环境。例如，清单 4.1 中的 rnn.c 程序通过 gcc 进行编译并执行时，能够学习并收敛。将相同的 rnn.c 程序用 Visual Studio 编译时，有时会发生学习不收敛的情形。这一无法避免的结果是由随机数的实现方法等各处理系统间的不一致所导致的。

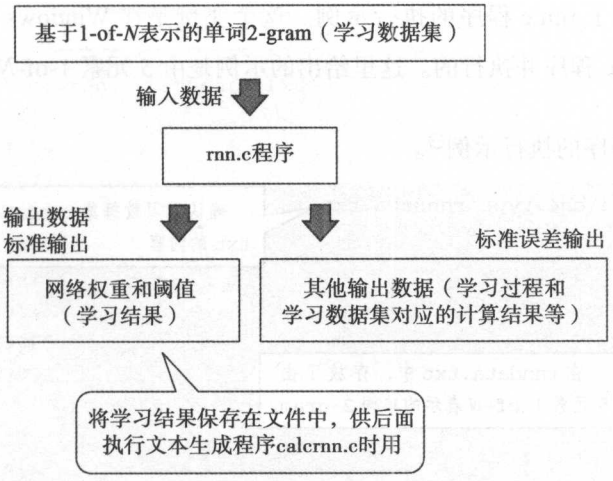


图 4.12　rnn.c 程序的输入输出数据

当学习无法收敛时，将清单 4.1 中随机数的种子符号常数 SEED 设为其他值，有时学习能够很好地收敛。在清单 4.1 中，SEED 的值按如下方式定义：

```
23:#dedine SEED 65535              /* 随机数的种子 */
```

将这一行注释掉，去掉下一行（第 24 行）的注释符号，将 SEED 的值设为 7：

```
23:// #dedine SEED 65535           /* 随机数的种子 */
24:#dedine SEED 7                  /* 随机数的种子（其他值）*/
```

据此对 SEED 的值进行调整，学习结果会随之发生变化，根据情况的不同，学习可能会进行得非常顺利。也就是说，如果采用不同的随机数种子，学习情形会发生相应的变化。

在对随机数的初始值进行选择之外，如果对学习系数 ALPHA 的值进行变更，也可能使学习变得更加顺利。在清单 4.1 中，在第 22 行中将 ALPHA 的值按下述方法设为 10：

```
22:#define ALPHA 10               /* 学习系数 */
```

如果学习进行得不顺利，可以对这部分进行修改，例如试着将 ALPHA 的值改为 3 或者 1：

```
22:#define ALPHA 3                /* 修改学习系数为 3 的示例 */
```

实例 4.1 给出了 rnn.c 程序的执行示例。这个实例是在 Windows 下的命令行窗口中利用 gcc 编译 rnn.c 程序并执行的。这里给出的示例是由 5 元素 1-of-*N* 表示的词链。

实例 4.1 rnn.c 程序的执行示例[⊖]。

```
C:\User\odaka\ch4>type rnndata.txt
1 0 0 0 0
1 0 0 0 0

1 0 0 0 0
0 1 0 0 0

0 1 0 0 0
0 0 1 0 0

0 0 1 0 0
0 0 0 1 0

0 0 0 1 0
0 0 0 0 1

0 0 0 0 1
0 0 0 0 1

0 0 0 0 1
0 0 1 0 0

0 0 1 0 0
0 1 0 0 0

0 1 0 0 0
0 0 1 0 0

0 0 1 0 0
0 0 0 1 0

1 1 1 1 1
1 1 1 1 1
```

确认学习数据集 rnndata.txt 的内容

在 rnndata.txt 中，存放了由 5 元素 1-of-*N* 表示的单词 2-gram

00100 → 00010

00100 → 01000，和上例不同的词链

⊖ 按书中提示将 rnn.c 中随机数种子修改为 7，学习系数修改为 3 后，在 Microsoft Visual Studio Professional 2013 下编译执行该程序，和原书在 gcc 下编译执行的结果略有不同（原书迭代约 18 万次后收敛），这里报告的是译者执行后的结果。

基于 rnn.c 程序的学习

```
C:\User\odaka\ch4>rnn1 < rnndata.txt
学习数据的个数: 11
5       12.937175
10      13.041372
15      13.067110
20      12.883410
25      12.521925
30      12.120113
35      11.783769
40      11.467800
(下面继续输出)
72555   0.011188
72560   0.010816
72565   0.010495
72570   0.010216
72575   0.009973
```

迭代约 7 万 2 千次后学习结束

输出作为学习结果的网络权重

```
11.831 -13.514  3.041  -7.169 -22.377  3.317 -26.812  4.722 7.815  11.885 -6.864
 4.660   5.930  7.467  -7.570  -4.487  4.049  10.647  1.281 0.158 -13.665 -1.899
 1.450  -0.242  0.960  -0.093   0.328  1.150   1.813  1.744 1.688   1.420 -3.166
-4.045  -4.696 -2.225  -4.849  -3.178  3.132   2.856  3.234 3.427 -0.455  -3.478
 9.985  -7.378 12.041   5.111  -9.336  3.290  -6.656 -3.363 6.867   0.660  3.795
 10.199  9.051  1.548 -18.510  -6.623 -0.843
  9.168  6.238 -8.513 -12.706  18.828  8.408
-10.473  7.951  6.053 -13.220 -14.203 -7.003
 -9.807  1.958  0.813  -6.224   7.650 -0.046
  0.250 -12.420 7.815 -11.173   1.347 -7.781
```

学习数据集对应的网络输出

```
0:
1.000 0.000 0.000 0.000 0.000 0.000 0.998 1.000 0.002 0.985
1.000 0.000 0.000 0.000 0.000
0.948 0.000 0.038 0.000 0.001
1:
1.000 0.000 0.000 0.000 0.000 1.000 0.993 1.000 0.994 0.041
0.000 1.000 0.000 0.000 0.000
0.030 0.990 0.000 0.004 0.002
2:
0.000 1.000 0.000 0.000 0.000 1.000 1.000 1.000 1.000 0.998

0.000 0.000 1.000 0.000 0.000
0.001 0.000 1.000 0.031 0.000
3:
0.000 0.000 1.000 0.000 0.000 0.003 1.000 1.000 1.000 0.000
```

大体上得到了和教师数据相对应的输出结果

00100 → 0.000 0.010 0.002 0.983 0.001

```
0.000 0.000 0.000 1.000 0.000
0.000 0.010 0.002 0.983 0.001
4:
0.000 0.000 0.000 1.000 0.000 0.013 1.000 1.000 1.000 0.994
0.000 0.000 0.000 0.000 1.000
0.000 0.000 0.014 0.016 0.991
5:
0.000 0.000 0.000 0.000 1.000 0.062 0.001 1.000 1.000 0.242
0.000 0.000 0.000 0.000 1.000
0.000 0.000 0.004 0.000 0.988
6:
0.000 0.000 0.000 0.000 1.000 0.521 0.015 0.999 0.999 0.000
0.000 0.000 1.000 0.000 0.000
0.000 0.000 0.984 0.004 0.007
7:
0.000 0.000 1.000 0.000 0.000 0.161 0.753 0.999 1.000 0.000
0.000 1.000 0.000 0.000 0.000
0.017 0.983 0.000 0.007 0.002
8:
0.000 1.000 0.000 0.000 0.000 0.942 1.000 1.000 1.000 0.999
0.000 0.000 1.000 0.000 0.000
0.001 0.000 1.000 0.031 0.000
9:
0.000 0.000 1.000 0.000 0.000 0.003 1.000 1.000 1.000 0.000
0.000 0.000 0.000 1.000 0.000
0.000 0.010 0.002 0.983 0.001
10:
1.000 1.000 1.000 1.000 1.000 0.013 1.000 1.000 1.000 0.994
1.000 1.000 1.000 1.000 1.000
0.992 1.000 0.999 1.000 0.989

C:\Users\odaka\ch4>
```

00100→0.017 0.983 0.000
0.007 0.002

存放学习数据集的 rnndata.txt 文件中保存了 11 个单词 2-gram，其中，也包括依存于上下文关系的词链变化的例子。

例如，学习数据集第 4 个例子中单词（0 0 1 0 0）的下一个单词是由

0 0 1 0 0 → 0 0 0 1 0

得到，即单词（0 0 1 0 0）的下一个单词是（0 0 0 1 0）。而第 8 个学习数据，对于同样的单词（0 0 1 0 0），有

```
0 0 1 0 0 → 0 1 0 0 0
```

也就是说，和该单词连接在一起的是和第 4 个学习数据不一样的单词（0 1 0 0 0）。这样一来，对于存放学习数据集的 rnndata.txt 文件而言，如果不考虑上下文关系，就无法学习到单词的词链关系。

rnn.c 程序使用存放在 rnndata.txt 中的学习数据集来进行学习，在实例 4.1 中，迭代了 7 万 2 千次左右学习结束。检查对学习数据集的学习结果可以发现，对于同一个单词（0 0 1 0 0），程序给出了不同的输出。下面给出具体的说明。

rnn.c 程序对学习数据集给出其计算结果时，网络输出了输入单词的数据、与之对应的教师数据以及循环神经网络的输出数据，而且一并输出了输入数据和上一次迭代的中间层输出数据。

例如，对于第 4 个学习数据的计算结果如下所示（译者注：原书有误，程序输出时序号是从 0 开始的，因此这个结果是第 4 个学习数据）：

```
3:
0.000 0.000 1.000 0.000 0.000 0.003 1.000 1.000 1.000 0.000
0.000 0.000 0.000 1.000 0.000
0.000 0.010 0.002 0.983 0.001
```

这一结果中，从第一行的第 1～5 个数值表示的是输入单词（0 0 1 0 0），第 6～10 个数值表示的是上一次迭代中间层的输出。第二行中表示的是教师数据（0 0 0 1 0），第三行表示的是网络的输出值。网络的输出是（0.000 0.010 0.002 0.983 0.001），虽然有一定的误差，但基本上和教师数据是一致的。

再如，对第 8 个学习数据的计算结果如下所示（译者注：原书有误，程序输出时序号是从 0 开始的，因此这个结果是第 8 个学习数据）：

```
7:
0.000 0.000 1.000 0.000 0.000 0.161 0.753 0.999 1.000 0.000
0.000 1.000 0.000 0.000 0.000
0.017 0.983 0.000 0.007 0.002
```

和前面的例子相同，输入单词（0 0 1 0 0）的教师数据是（0 1 0 0 0），上面给出了

其学习结果。网络的输出是（0.017 0.983 0.000 0.007 0.002），这一结果也有一定的误差，但大致上和教师数据是一致的。顺便说一下，表示上一次迭代中间层输出的第 6～10 个数值和第 3 个学习数据有非常大的差异，因此，对于同样的输入单词，循环神经网络可得到相异的输出结果。

比较上述两个结果可以发现，由于上下文关系，对于相同的输入单词，循环神经网络的输出结果是不同的。如果对于相同的学习数据集采用没有反馈机制的神经网络，对应于某单词的输出会固定为同一个单词，而无法考虑上下文关系以生成相异的单词。在实例 4.1 中，对于单词（0 0 1 0 0），由于上下文关系，rnn.c 生成了单词（0 0 0 1 0）和单词（0 1 0 0 0）。因此，rnn.c 是能够考虑上下文关系进行学习的神经网络程序。

4.3 基于 RNN 的文本生成

4.3.1 基于 RNN 的文本生成框架

rnn.c 程序能获得生成词链的知识，考虑应用这些知识生成新文本的方法。为此，采用 rnn.c 程序的学习结果来创建从适当的单词开始生成词链的 calcrnn.c 程序。

calcrnn.c 程序会读取 rnn.c 程序学习结果中的连接权重数据，在命令行参数中给出开始单词的序号，从该单词开始生成单词序列。图 4.13 给出了 calcrnn.c 程序的使用方法。

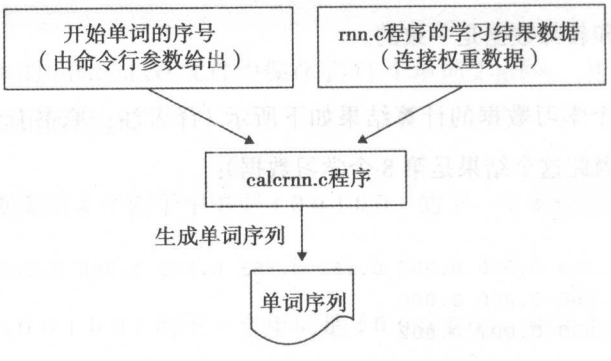


图 4.13 calcrnn.c 程序的使用方法

图 4.13 以 rnn.c 程序的学习为前提，rnn.c 学习结束后会输出其网络的权重数据。calcrnn.c 程序使用这一权重数据来生成单词的词链。

在执行 calcrnn.c 程序的命令行参数中给出作为词链开头的开始单词的序号。calcrnn.c 程序采用学习得到的权重，用循环神经网络按顺序生成单词词链。生成的单词词链个数由 calcrnn.c 程序内的符号常数来指定。

清单 4.2 给出了基于以上前提所创建的 calcrnn.c 程序。

清单 4.2 calcrnn.c 程序。

```
/*******************************************************/
/*                    calcrnn.c                        */
/*                    循环神经网络                      */
/*          学习完成后的神经网络的计算                  */
/* 使用方法                                            */
/*  \Users\odaka\ch4>calcrnn 2 < data.txt              */
/*  将开始单词的序号作为命令行的参数给出                */
/*******************************************************/

/* 和 Visual Studio 的互换性保证 */
#define _CRT_SECURE_NO_WARNINGS

/* include 头文件 */
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

/* 符号常数的定义 */
#define INPUTNO 5     /* 输入的元素数 */
#define HIDDENNO 5    /* 中间层神经元数 */
#define OUTPUTNO 5    /* 输出层神经元数 */
#define WORDLEN 50    /* 单词词链长度 */

/* 函数原型的声明 */
double f(double u) ; /* 传递函数 (sigmoid 函数) */
void print(double wh[HIDDENNO][INPUTNO+1+HIDDENNO]
          ,double wo[OUTPUTNO][HIDDENNO+1]) ; /* 输出结果 */
double forward(double wh[HIDDENNO][INPUTNO+1+HIDDENNO]
          ,double wo[HIDDENNO+1],double hi[]
          ,double e[]) ; /* 前向计算 */
```

```
31 void readwh(double wh[HIDDENNO][INPUTNO+1+HIDDENNO]) ;
32                              /* 读入中间层权重 */
33 void readwo(double wo[OUTPUTNO][HIDDENNO+1]) ;
34                              /* 读入输出层权重 */
35 void  putword(double inputdata[]) ;
36                              /* 仅将单词部分用 1-of-N 表示 */
37
38 /********************/
39 /*     main() 函数    */
40 /********************/
41 int main(int argc,char *argv[])
42 {
43  double wh[HIDDENNO][INPUTNO+1+HIDDENNO] ; /* 中间层权重 */
44  double wo[OUTPUTNO][HIDDENNO+1] ;          /* 输出层权重 */
45  double hi[HIDDENNO+1]={0} ;                /* 中间层的输出 */
46  double o[OUTPUTNO] ;                       /* 输出 */
47  double inputdata[INPUTNO+HIDDENNO]={0} ;   /* 输入 */
48  int i,j ;                                  /* 循环控制用 */
49  int startno ;                              /* 开始单词的序号 */
50
51  /* 指定开始单词 */
52  if(argc<2){
53   startno=0 ;/* 没有指定时从序号 0 开始 */
54  }
55  else{
56   startno=atoi(argv[1]) ;
57   if((startno<0) || (startno>=INPUTNO)){
58    /* 指定开始单词错误 */
59    fprintf(stderr," 開始単語の指定値 (%d) が間違っています \n",startno) ;
60    exit(1) ;
61   }
62  }
63  inputdata[startno]=1.0 ;
64
65  /* 读入权重 */
66  readwh(wh) ;          /* 读入中间层权重 */
67  readwo(wo) ;          /* 读入输出层权重 */
68
69  /* 文本生成 */
70  for(i=0;i<WORDLEN;++i){
71   /* 表示输入数据 */
72   /* 有必要的话，去掉下面三行的注释符 */
73 //  for(j=0;j<INPUTNO+HIDDENNO;++j)
74 //   printf("%.3lf ",inputdata[j]) ;
```

```
75 //  printf("\n") ;
76   /* 将单词部分用 1-of-N 表示 */
77   putword(inputdata) ;
78   /* 前向计算 */
79   for(j=0;j<OUTPUTNO;++j)
80    o[j]=forward(wh,wo[j],hi,inputdata) ;
81   /* 将前一次的输出设定为输入 */
82   for(j=0;j<HIDDENNO;++j)
83     inputdata[j]=o[j] ;
84   /* 前一次中间层的输出追加到输入中 */
85   for(j=0;j<HIDDENNO;++j)
86     inputdata[INPUTNO+j]=hi[j] ;
87  }
88
89  return 0 ;
90 }
91
92 /*********************************/
93 /*   putword() 函数                */
94 /* 将单词部分用 1-of-N 表示          */
95 /*********************************/
96 void  putword(double inputdata[])
97 {
98  int i ;                        /* 循环控制用 */
99  int maxindex=0 ;               /* 最大元素值的序号 */
100  double max=inputdata[0] ;     /* 最大值 */
101
102  /* 调查最大的元素 */
103 for(i=1;i<INPUTNO;++i)
104  if(max<inputdata[i]){
105   max=inputdata[i] ;
106   maxindex=i ;
107  }
108  /* 输出 1-of-N 表示的单词 */
109  for(i=0;i<INPUTNO;++i){
110   if(i==maxindex) printf("1 ") ;
111   else printf("0 ") ;
112  }
113  printf("\n") ;
114
115 }
116
117 /**********************/
118 /*  forward() 函数      */
```

```
/*  前向计算            */
/**********************/
double forward(double wh[HIDDENNO][INPUTNO+1+HIDDENNO]
 ,double wo[HIDDENNO+1],double hi[],double e[])
{
 int i,j ;              /* 最大値要素の添え字番号 */
 double u ;             /* 计算加权和 */
 double o ;             /* 输出的计算 */

 /*hi 的计算 */
 for(i=0;i<HIDDENNO;++i){
  u=0 ;                 /* 求得加权和 */
  for(j=0;j<INPUTNO+HIDDENNO;++j)
   u+=e[j]*wh[i][j] ;
  u-=wh[i][j] ;         /* 处理阈值 */
  hi[i]=f(u) ;
 }
 /* 计算输出 o*/
 o=0 ;
 for(i=0;i<HIDDENNO;++i)
  o+=hi[i]*wo[i] ;
 o-=wo[i] ;             /* 处理阈值 */

 return f(o) ;
}

/*************************/
/*    readwh() 函数       */
/* 读入中间层权重          */
/*************************/
void readwh(double wh[HIDDENNO][INPUTNO+1+HIDDENNO])
{
 int i,j ;              /* 循环控制用 */

  for(i=0;i<HIDDENNO;++i){
  for(j=0;j<INPUTNO+1+HIDDENNO;++j)
   scanf("%lf",&(wh[i][j])) ;
 }
}

/*************************/
/*    readwo() 函数       */
/*  读入输出层权重        */
/*************************/
```

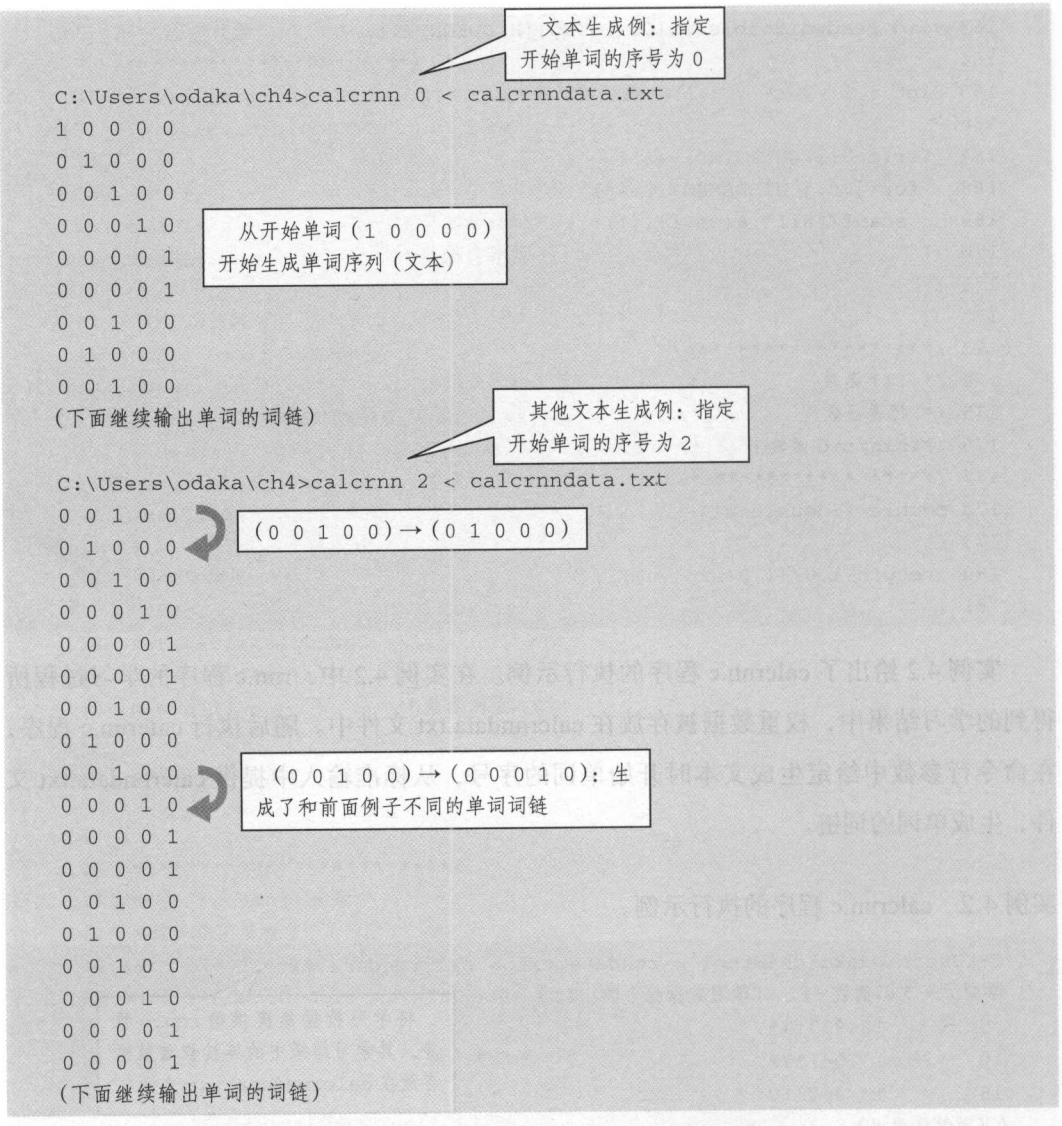
```
163 void readwo(double wo[OUTPUTNO][HIDDENNO+1])
164 {
165  int i,j ;            /* 循环控制用 */
166
167  for(i=0;i<OUTPUTNO;++i){
168   for(j=0;j<HIDDENNO+1;++j)
169    scanf("%lf",&(wo[i][j])) ;
170  }
171 }
172
173 /*******************/
174 /* f() 函数          */
175 /* 传递函数          */
176 /*(sigmoid 函数)    */
177 /*******************/
178 double f(double u)
179 {
180  return 1.0/(1.0+exp(-u)) ;
181 }
```

实例 4.2 给出了 calcrnn.c 程序的执行示例。在实例 4.2 中，rnn.c 程序的学习过程所得到的学习结果中，权重数据被存放在 calcrnndata.txt 文件中。随后执行 calcrnn.c 程序，在命令行参数中给定生成文本时开始单词的序号，从标准输入中提供 calcrnndata.txt 文件，生成单词的词链。

实例 4.2 calcrnn.c 程序的执行示例。

```
C:\Users\odaka\ch4>rnn1 < rnndata.txt > calcrnndata.txt
学習データの個数 :11 (学习数据的个数: 11)
 5        12.937175
10        13.041372
15        13.067110
(下面继续输出)
72565    0.010495
72570    0.010216
72575    0.009973
0:
1.000 0.000 0.000 0.000 0.000 0.000 0.998 1.000 0.002 0.985
1.000 0.000 0.000 0.000 0.000
0.948 0.000 0.038 0.000 0.001
(下面继续输出对学习数据集的计算结果)
```

将学习数据集提供给 rnn.c 程序，其学习结果中的连接权重数据存放在 calcrnndata.txt 文件中



在实例4.2中，给定两个相异的开始单词后，calcrnn.c程序给出了相应的输出结果。一点也不奇怪，在这两种情况下生成了相异的单词序列。而且，在生成文本的过程中，对应于同一个单词，根据上下文的不同，会生成不同的下一个单词。

4.3.2 文本生成实验的实例

最后，给出应用循环神经网络来生成文本的执行示例。下面选取本书2.1.3节中含有

45 个单词的例题作为学习对象来生成文本。

在实例 4.3 中，首先给出了从要学习文章中创建 1-of-*N* 表示的学习数据的步骤。该处理和 2.1.3 节中给出的步骤是相同的。也就是说，使用第 2 章中介绍的 makew1gram.c 程序和 makevec.c 程序将分析对象 text3.txt 变换为 1-of-*N* 表示的单词序列。在这一过程中，将所有单词存放在 voc.txt 文件中。

实例 4.3 从要学习的文本中创建 1-of-*N* 表示的单词序列数据的步骤。

```
C:\Users\odaka\ch4>type text3.txt
  自然言語処理の技術を用いると、文書の要約や、文書どうしの類似性を評価することが
できます。文書要約においては、ある文書に含まれ用語のうちから文書の特徴を表す重要
語を抽出したり、文書を表現する要約文を作成する技術が利用されています。また、こう
した技術を用いて、複数の文書どうしの類似性を数値で評価する手法が提案されています。

C:\Users\odaka\ch4>makew1gram < text3.txt > w1gram.txt

C:\Users\odaka\ch4>makevec > makevecout.txt
単語数 45

C:\Users\odaka\ch4>type makevecout.txt
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0
(下面显示文件的内容)

C:\Users\odaka\ch4>
```

确认分析对象 text3.txt 的内容

用 makew1gram.c 程序变换为单词 1-gram

用 makevec.c 程序变换为 1-of-N 表示的单词序列

makevec.c 程序的输出结果（1-of-*N* 表示的单词序列）

下面从所创建的 makevecout.txt 文件中生成单词 2-gram。为此，用 dupline.c 程序将单词序列变换为 2-gram。dupline.c 程序将单词序列第 2 行以后的每一行都复制一下，生成 1-of-*N* 表示的单词 2-gram，这一单词 2-gram 作为学习数据集使用。由 dupline.c 程序生成单词 2-gram 的示例在实例 4.4 中给出，dupline.c 程序的源代码在清单 4.3 中给出。

实例 4.4 由 1-of-*N* 表示的单词序列生成单词 2-gram 的示例。

```
C:\Users\odaka\ch4>dupline < makevecout.txt > rnndata.txt
                          用 dupline.c 程序将单词序列变换为 2-gram

C:\Users\odaka\ch4>type rnndata.txt
1 0 0 0
```

```
23  char line[MAXSIZE] ;/* 输入行 */
24
25  /* 读入最初行并原样输出 */
26  fgets(line,MAXSIZE,stdin) ;
27  printf("%s",line) ;
28
29  /* 复制第 2 行之后的行 */
30  while(fgets(line,MAXSIZE,stdin)!=NULL){
31   printf("%s",line) ;
32   printf("%s",line) ;
33  }
34
35  return 0 ;
36 }
```

这样就得到了存放学习数据集的 rnndata.txt 文件。下面用 rnn.c 程序来进行神经网络的学习。为了和学习数据集的大小相吻合，需要对 rnn.c 程序的一部分进行修改。具体而言，如清单 4.4 所示，将神经网络各层的神经元数目和问题的大小相一致起来进行修改。由于这里要处理的是由 45 种单词组成的学习数据，因此网络各层的神经元数应变更为 45。另外，为了和网络的变更相吻合，对学习系数 ALPHA 和误差的上限值 LIMIT 也要按照以下方式进行调整。在后面的 calcrnn.c 程序中也要同样实施这里的变更。

程序清单 4.4 和问题的大小相吻合，修改符号常数（rnn.c 程序及 calcrnn.c 程序）。

变更前

```
18: /* 符号常数的定义 */
19: #define INPUTNO 5              /* 输入的元素数 */
20: #define HIDDENNO 5             /* 中间层神经元数 */
21: #define OUTPUTNO 5             /* 输出层神经元数 */
22: #define ALPHA  10              /* 学习系数 */
23: #define SEED 65535             /* 随机数的种子 */
24: //#define SEED 7               /* 随机数的种子（其他值）*/
25: #define MAXINPUTNO 100         /* 学习数据的最大个数 */
26: #define BIGNUM 100             /* 误差的初始值 */
27: #define LIMIT 0.01             /* 误差的上限值 */
```

变更后

```
18: /* 符号常数的定义 */
19: #define INPUTNO 45             /* 输入的元素数 */
```

```
20: #define HIDDENNO 45          /* 中间层神经元数 */
21: #define OUTPUTNO 45          /* 输出层神经元数 */
22: #define ALPHA  0.2           /* 学习系数 */
23: #define SEED 65535           /* 随机数的种子 */
24: //#define SEED 7             /* 随机数的种子（其他值）*/
25: #define MAXINPUTNO 100       /* 学习数据的最大个数 */
26: #define BIGNUM 100           /* 误差的初始值 */
27: #define LIMIT 2              /* 误差的上限值 */
```

下面基于 rnn.c 程序来学习。实例 4.5 中给出了学习过程。

实例 4.5 rnn.c 程序的学习过程。

```
C:\Users\odaka\ch4>rnn < rnndata.txt > calcrnndata.txt

学习数据的个数: 76
 45       254.882615
 90       130.521052
135        74.495603
180        73.648223
(下面继续输出)
99810     1.986350
0:
1.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000
(下面输出对学习数据集的计算结果)
C:\Users\odaka\ch4>
```

基于 rnn.c 程序学习
学习结果中的权重系数存放在 calcrnndata.txt 文件中

到实例 4.5 为止的处理备齐了提供给 calcrnn.c 程序的数据。有了以上的准备工作，可以用 calcrnn.c 生成单词的词链。实例 4.6 给出了生成新单词序列的步骤。

实例 4.6 calcrnn.c 程序生成单词序列。

从单词序号 8 开始生成词链，通过 makes.c 程序变换为自然语言表示

执行 make.c 程序时需要 voc.txt 文件

```
C:\Users\odaka\ch4>calcrnn1 8 < calcrnndata.txt | makes
単語数 45
文書どうしの類似性を数値で評価する手法が利用されています。またのこうしたを技術を
利用されています評価する手法こうした評価またをにで評価する手法が利用されています。
の、こうしたををが利用されていますを用するにおいてはが
C:\Users\odaka\ch4>calcrnn1 18 < calcrnndata.txt | makes
```

生成的文本

从单词序号 18 开始生成词链

```
単語数 45
ある文書を表現する要約文を作成する技術が利用されています。また、こうした技術を用
いて、複数のこうしたこうした表こうしたこうした、こうしたを数値で評価する利用が利
用されていますを用がこうした複数技術がにされていますを
C:\Users\odaka\ch4>calcrnn1 32 < calcrnndata.txt | makes
単語数 45
```

从单词序号 32 开始生成词链

```
要約文を表現する要約文がまれする、文書をどうしのを作成を作成する技術が利用されて
います。またするこうしたがこうしたする利用されていますをどうしのをのをこうしたを
にが利用されていますをのするこうしたをこうしたするにこうした
C:\Users\odaka\ch4>calcrnn1 40 < calcrnndata.txt | makes
単語数 45
```

从单词序号 40 开始生成词链

```
複数のを技術を利用されています評価されていますのの特徴またをにで評価する手法が利用されています。
の文書要約こうしたこうしたをが利用されていますをまたする数値がこうしたするをこうしたをどうしの
を用で評価する技術が利用
C:\Users\odaka\ch4>
```

在实例 4.6 中，应用了 calcrnn.c 程序来生成单词序列。calcrnn.c 程序从命令行参数中获取开始单词的序号。实例 4.6 在执行 calcrnn.c 程序时，通过命令行参数给出了多个不同的开始序号，且按标准输入方式给出了生成文本所需的 calcrnndata.txt 的数据。

calcrnn.c 程序的输出以管道（pipe）为媒介提供给 makes.c 程序。makes.c 程序从标准输入中获取 1-of-*N* 表示的单词序列，将其变换为一般的日语输出。如第 2 章所介绍，执行 makes.c 程序时需要 voc.txt 文件。

在实例 4.6 中，对于 4 种不同的开始单词，生成了相应的单词序列。将各个输出例子的开头部分集中起来，可得到下面文本的集合。

文書どうしの類似性を数値で評価する手法が利用されています。（用数值方法来评估文本之间的相似性。）

ある文書を表現する要約文を作成する技術が利用されています。（使用了表示某文本摘要文本的生成技术。）

要約文を表現する要約文がまれする、文書をどうしのを作成を作成する技術が利用されています[⊖]。

⊖ 原文是不符合日语语法的怪怪的日语文本。——译者注

複数のを技術を利用されています評価されていますのの特徴またをにで評価する手法が利用されています[⊖]。

随着应用领域和用途的不同，对这种自动生成的文本的评价也不同。但有一点是显而易见的，即生成文本时采用的语法规则和词的用法仿佛有某种癖好，能生成不可思议的文本。

⊖ 原文是不符合日语语法的怪怪的日语文本。——译者注

APPENDIX A

附录 A

将行的重复次数添加到行首的程序 uniqc.c

正如第 2 章所介绍的，uniqc.c 程序删除重复的行，将行的重复次数添加到行首，清单 A 给出了 uniqc.c 程序的源代码。

清单 A uniqc.c 程序。

```
/*********************************************/
/*            uniqc.c                        */
/*  对重复行计数                             */
/*  删除重复行，表示重复次数                 */
/* 使用方法                                  */
/*C:\Users\odaka\ch2>uniqc < text.txt         */
/*********************************************/

/* 和 Visual Studio 的互换性保证 */
#define _CRT_SECURE_NO_WARNINGS

/*include 头文件 */
#include <stdio.h>
#include <stdlib.h>
#include<string.h>

/* 符号常数的定义 */
#define MAXLINE 65535

/****************/
/*  main() 函数  */
/****************/
```

```
23 int main()
24 {
25  char newline[MAXLINE] ;      /* 输入行 */
26  char oldline[MAXLINE] ;      /* 前一行 */
27  int count=1 ;                /* 重复次数 */
28
29  /* 读入文本 */
30  fgets(oldline,MAXLINE,stdin) ;
31  while(fgets(newline,MAXLINE,stdin)!=NULL){
32   if(strcmp(newline,oldline)==0) ++count ; /* 相同的行 */
33   else{                                    /* 不同的行 */
34    printf("%d\t%s",count,oldline);
35    count=1 ;
36    strcpy(oldline,newline) ;
37   }
38  }
39  printf("%d\t%s",count,oldline);
40
41  return 0 ;
42 }
```

APPENDIX B

附录 B

按照行首的数值对行进行排序的程序 sortn.c

正如第 2 章所介绍的，sortn.c 程序按照行首的数值对行进行排序，清单 B 给出了 sortn.c 程序的源代码。

清单 B sortn.c 程序。

```
/*********************************************/
/*            sortn.c                        */
/*  根据行首的数值按顺序排列                 */
/* 使用方法                                   */
/*C:\Users\odaka\ch2>sortn < text.txt         */
/*********************************************/

/* 和 Visual Studio 的互换性保证 */
#define _CRT_SECURE_NO_WARNINGS

/*include 头文件 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/* 符号常数的定义 */
#define LINESIZE 256          /* 一行字节数的上限 */
#define MAX 65536*3           /* 行数的上限 */

/* 函数原型的声明 */
int cmpdata(const char *a,const char *b) ; /* 比较函数 */

/* 外部变量 */
```

```
24 char lines[MAX][LINESIZE] ; /*处理对象文本*/
25
26 /****************/
27 /* main()函数*/
28 /****************/
29 int main()
30 {
31  char buffer[LINESIZE] ;      /*读入缓存*/
32  int pos=0 ;                  /*读入行数的计数*/
33  int i ;
34
35  /*读入文本*/
36  while(fgets(buffer,LINESIZE,stdin)!=NULL){
37    strcpy(lines[pos],buffer) ;
38    if((++pos)>=MAX){
39     fprintf(stderr,
40       "ファイルサイズの上限を超えました\n") ;
41     exit(1);
42    }
43   }
44  /*按序排列*/
45  qsort(lines,pos,LINESIZE,
46    (int (*)(const void *,const void *))cmpdata) ;
47  /*出力*/
48  for(i=0;i<pos;++i)
49   printf("%s",lines[i]) ;
50
51  return 0;
52 }
53
54 /*******************************/
55 /*  cmpdata()函数               */
56 /*  比较函数                    */
57 /*******************************/
58 int cmpdata(const char *a,const char *b)
59 {
60  int inta,intb ;
61
62  inta=atoi(a) ;
63  intb=atoi(b) ;
64  if(inta>intb) return -1;          /*如果第一参数大，则返回-1*/
65  else if(inta<intb) return 1 ;     /*如果第二参数大，则返回1*/
66
67  return 0 ;/*其他*/
68 }
```

APPENDIX C

附录 C

全连接型神经网络的程序 bp.c

正如第 3 章所介绍的，bp.c 程序使用反向传播进行神经网络的学习，清单 C 给出了 bp.c 程序的源代码。

清单 C　bp.c 程序。

```
/*************************************************************/
/*                          bp.c                             */
/*  通过反向传播进行神经网络的学习                           */
/* 使用方法                                                  */
/*  \Users\odaka\ch3>bp < data.txt > result.txt              */
/*  输出误差的变化、作为学习结果的连接系数等                 */
/*************************************************************/

/* 和 Visual Studio 的互换性保证 */
#define _CRT_SECURE_NO_WARNINGS

/* include 头文件 */
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

/* 符号常数的定义 */
#define INPUTNO 2              /* 输入层神经元数 */
#define HIDDENNO 2             /* 中间层神经元数 */
#define ALPHA  10              /* 学习系数 */
#define SEED 65535             /* 随机数种子 */
#define MAXINPUTNO 100         /* 学习数据的最大个数 */
```

```
#define BIGNUM 100     /* 误差的初始值 */
#define LIMIT 0.001     /* 误差的上限值 */

/* 函数原型的声明 */
double fs(double u) ;                          /* 传递函数 (sigmoid 函数) */
void initwh(double wh[HIDDENNO][INPUTNO+1]); /* 中间层权重的初始化 */
void initwo(double wo[HIDDENNO+1]) ;          /* 输出层权重初始化 */
double drnd(void) ;                            /* 生成随机数 */
void print(double wh[HIDDENNO][INPUTNO+1]
          ,double wo[HIDDENNO+1]) ;           /* 输出结果 */
double forward(double wh[HIDDENNO][INPUTNO+1]
         ,double wo[HIDDENNO+1],double hi[]
         ,double e[INPUTNO+1]) ;               /* 前向计算 */
void olearn(double wo[HIDDENNO+1],double hi[]
         ,double e[INPUTNO+1],double o) ;      /* 调整输出层权重 */
int getdata(double e[][INPUTNO+1]) ;           /* 读入学习数据 */
void hlearn(double wh[HIDDENNO][INPUTNO+1]
         ,double wo[HIDDENNO+1],double hi[]
         ,double e[INPUTNO+1],double o);       /* 调整中间层权重 */

/********************/
/*    main() 函数    */
/********************/
int main()
{
 double wh[HIDDENNO][INPUTNO+1] ;    /* 中间层权重 */
 double wo[HIDDENNO+1] ;             /* 输出层权重 */
 double e[MAXINPUTNO][INPUTNO+1] ;   /* 学习数据集 */
 double hi[HIDDENNO+1] ;             /* 中间层的输出 */
 double o ;                          /* 输出 */
 double err=BIGNUM ;                 /* 误差 */
 int i,j ;                           /* 循环控制变量 */
 int n_of_e ;                        /* 学习数据的个数 */
 int count=0 ;                       /* 循环次数的计数 */

 /* 随机数的初始化 */
 srand(SEED) ;

 /* 权重的初始化 */
 initwh(wh) ;                        /* 中间层权重的初始化 */
 initwo(wo) ;                        /* 输出层权重的初始化 */
 print(wh,wo) ;                      /* 输出权重 */

 /* 读入学习数据 */
```

```
67  n_of_e=getdata(e) ;
68  printf("学習データの個数 :%d\n",n_of_e) ;
69
70  /*学习*/
71  while(err>LIMIT){/*迭代直到误差收敛*/
72   err=0.0 ;
73   for(j=0;j<n_of_e;++j){
74    /*前向计算*/
75    o=forward(wh,wo,hi,e[j]) ;
76    /*调整输出层权重*/
77    olearn(wo,hi,e[j],o) ;
78    /*调整中间层权重*/
79    hlearn(wh,wo,hi,e[j],o) ;
80    /*误差的累积*/
81    err+=(o-e[j][INPUTNO])*(o-e[j][INPUTNO]) ;
82   }
83   ++count ;
84   /*误差的输出*/
85   fprintf(stderr,"%d\t%lf\n",count,err) ;
86  }/*学习结束*/
87
88  /*连接权重的输出*/
89  print(wh,wo) ;
90
91  /*对于学习数据的计算结果的输出*/
92  for(i=0;i<n_of_e;++i){
93   printf("%d ",i) ;
94   for(j=0;j<INPUTNO+1;++j)
95    printf("%lf ",e[i][j]) ;
96   o=forward(wh,wo,hi,e[i]) ;
97   printf("%lf\n",o) ;
98  }
99
100 return 0 ;
101 }
102
103 /**********************/
104 /*  hlearn()函数      */
105 /*  学习中间层的权重  */
106 /**********************/
107 void hlearn(double wh[HIDDENNO][INPUTNO+1]
108     ,double wo[HIDDENNO+1]
109     ,double hi[],double e[INPUTNO+1],double o)
110 {
```

```
111  int i,j ;                              /* 循环控制变量 */
112  double dj ;                            /* 用于计算中间层的权重 */
113
114  for(j=0;j<HIDDENNO;++j){               /* 将中间层个神经元 j 作为对象 */
115   dj=hi[j]*(1-hi[j])*wo[j]*(e[INPUTNO]-o)*o*(1-o) ;
116   for(i=0;i<INPUTNO;++i)                /* 处理第 i 个权重 */
117    wh[j][i]+=ALPHA*e[i]*dj ;
118   wh[j][i]+=ALPHA*(-1.0)*dj ;           /* 阈值的学习 */
119  }
120 }
121
122 /**********************/
123 /*  getdata() 函数      */
124 /* 读入学习数据          */
125 /**********************/
126 int getdata(double e[][INPUTNO+1])
127 {
128  int n_of_e=0 ;                         /* 数据集的个数 */
129  int i=0 ;                              /* 循环控制变量 */
130
131  /* 输入数据 */
132  while(scanf("%lf",&e[n_of_e][i])!=EOF){
133   ++ i ;
134   if(i>INPUTNO){/* 下一个数据 */
135    i=0 ;
136    ++n_of_e ;
137   }
138  }
139  return n_of_e ;
140 }
141
142 /**********************/
143 /*  olearn() 函数       */
144 /*  学习输出层权重       */
145 /**********************/
146 void olearn(double wo[HIDDENNO+1]
147     ,double hi[],double e[INPUTNO+1],double o)
148 {
149  int i ;                        /* 循环控制变量 */
150  double d ;                     /* 用于计算权重 */
151
152  d=(e[INPUTNO]-o)*o*(1-o) ; /* 误差的计算 */
153  for(i=0;i<HIDDENNO;++i){
154   wo[i]+=ALPHA*hi[i]*d ;     /* 学习权重 */
```

```
155  }
156  wo[i]+=ALPHA*(-1.0)*d ;/* 学习阈值 */
157
158 }
159
160 /**********************/
161 /*  forward() 函数     */
162 /*  前向计算           */
163 /**********************/
164 double forward(double wh[HIDDENNO][INPUTNO+1]
165  ,double wo[HIDDENNO+1],double hi[],double e[INPUTNO+1])
166 {
167  int i,j ;    /* 循环控制变量 */
168  double u ;   /* 计算加权和 */
169  double o ;   /* 计算输出 */
170
171  /*hi 的计算 */
172  for(i=0;i<HIDDENNO;++i){
173   u=0 ;              /* 求得加权和 */
174   for(j=0;j<INPUTNO;++j)
175    u+=e[j]*wh[i][j] ;
176   u-=wh[i][j] ;      /* 处理阈值 */
177   hi[i]=fs(u) ;
178  }
179  /* 计算输出 o*/
180  o=0 ;
181  for(i=0;i<HIDDENNO;++i)
182   o+=hi[i]*wo[i] ;
183  o-=wo[i] ;          /* 处理阈值 */
184
185  return fs(o) ;
186 }
187
188 /**********************/
189 /*   print() 函数     */
190 /*   输出结果         */
191 /**********************/
192 void print(double wh[HIDDENNO][INPUTNO+1]
193           ,double wo[HIDDENNO+1])
194 {
195  int i,j ;           /* 循环控制变量 */
196
197  /* 输出中间层权重 */
198  for(i=0;i<HIDDENNO;++i)
```

```
199   for(j=0;j<INPUTNO+1;++j)
200    printf("%lf ",wh[i][j]) ;
201  printf("\n") ;
202  /*输出输出层权重*/
203  for(i=0;i<HIDDENNO+1;++i)
204   printf("%lf ",wo[i]) ;
205  printf("\n") ;
206 }
207
208 /**********************/
209 /*     initwh()函数     */
210 /*中间层权重的初始化     */
211 /**********************/
212 void initwh(double wh[HIDDENNO][INPUTNO+1])
213 {
214  int i,j ;            /*循环控制变量*/
215
216  /*由随机数确定权重*/
217  for(i=0;i<HIDDENNO;++i)
218   for(j=0;j<INPUTNO+1;++j)
219    wh[i][j]=drnd() ;
220 }
221
222 /**********************/
223 /*     initwo()函数     */
224 /*输出层权重的初始化     */
225 /**********************/
226 void initwo(double wo[HIDDENNO+1])
227 {
228  int i ;/*循环控制变量*/
229
230  /*由随机数确定权重*/
231  for(i=0;i<HIDDENNO+1;++i)
232    wo[i]=drnd() ;
233 }
234
235 /********************/
236 /* drnd()函数       */
237 /* 生成随机数       */
238 /********************/
239 double drnd(void)
240 {
241  double rndno ;       /*生成的随机数*/
242
```

```
243  while((rndno=(double)rand()/RAND_MAX)==1.0) ;
244  rndno=rndno*2-1 ;   /* 生成 -1 和 1 之间的随机数 */
245  return rndno;
246 }
247
248 /*******************/
249 /* fs() 函数        */
250 /* 传递函数         */
251 /*(sigmoid 函数)    */
252 /*******************/
253 double fs(double u)
254 {
255  return 1.0/(1.0+exp(-u)) ;
256 }
```

参考文献

1. 关于深度学习

[1] 人工知能学会（監修）、神嶌 敏弘（編）、深層学習、近代科学社、2015.

[2] 伊庭斉志、進化計算と深層学習—創発する知能—、オーム社、2015.

[3] 岡谷貴之、深層学習、講談社、2015.

[4] 小高知宏、機械学習と深層学習 —C言語によるシミュレーション—、オーム社、2016.

[5] 斎藤康毅、ゼロから作るDeep Learning —Pythonで学ぶディープラーニングの理論と実装、オライリージャパン、2016.

2. 关于自然语言处理

[1] 小高知宏、はじめてのAIプログラミング—C言語で作る人工知能と人工無能、オーム社、2006.

[2] Steven Bird他、入門 自然言語処理、オライリージャパン、2010.

译者简介

申富饶 南京大学计算机科学与技术系教授，博士生导师。主要研究方向包括神经网络、数据分析、机器人智能等，在国内外发表学术论文80余篇。

于 僳 毕业于日本德岛大学电气电子专业。现于日本从事机器人及电子制作教育教材开发、中小学生程序设计与机器人竞赛培训工作。

封面设计 锡 彬

自然言語処理と深層学習 C言語によるシミュレーション

本书详细介绍了将深度学习应用于自然语言处理的方法，并概述了自然语言处理的一般概念，通过具体实例说明了如何提取自然语言文本的特征以及如何考虑上下文关系来生成文本。书中自然语言文本的特征提取是通过卷积神经网络来实现的，而根据上下文关系来生成文本则利用了循环神经网络。这两个网络是深度学习领域中常用的基础技术。

本书通过实现C语言程序来具体讲解自然语言处理与深度学习的相关技术。本书给出的程序都能在普通个人电脑上执行。通过实际执行这些C语言程序，确认其运行过程，并根据需要对程序进行修改，读者能够更深刻地理解自然语言处理与深度学习技术。

上架指导：计算机/人工智能
ISBN 978-7-111-58657-9
9 787111 586579 >
定价：49.00元

投稿热线：(010) 88379604
客服热线：(010) 88379426 88361066
购书热线：(010) 68326294 88379649 68995259
华章网站：www.hzbook.com
网上购书：www.china-pub.com
数字阅读：www.hzmedia.com.cn